Dynamic execution unit management for high performance server system

Patent Number: EP0794490, A3
Publication date: 1997-09-10
Inventor(s): YEUNG LEO YUE TAK (US); SHARMA MOHAN (US)
Applicant(s): IBM (US)
Requested Patent: JP9265409
Application Number: EP19970300612 19970130
Priority Number(s): US19960613106 19960308
IPC Classification: G06F9/46; H04L29/06
EC Classification: H04L29/06, G06F9/46A2, G06F9/46R6
Equivalents: JP3229237B2, KR253930, US6182109
Cited patent(s): EP0694837; EP0384339; EP0413490; EP0666665; EP0473913

Abstract

A method, system and product for dynamically managing a pool of execution units in a server system, the pool devoted to a communication process between client and server processes. A minimum and a maximum number of execution units in the communication process pool is established. The minimum number of execution units is the number necessary to support a typical client load. The maximum number of execution units is an upper bound to support a peak client load without overloading the server system. As client requests for service are received by the server system, a number of determinations are made. It is determined whether assigning an execution unit to the request would bring a current number of execution units in the communication process pool over the maximum number of execution units. If so, the client request is rejected. It is determined whether assigning an execution unit to the request would bring the number of assigned execution units to a client task making the request over an allotted number of execution units for the client task. If so, the client request is rejected. The client request if the determinations are negative thereby assigning an execution unit in the communication process pool to the client request. The number of unused execution units in the communication pool is periodically reviewed to determine whether it should be increased or decreased to improve system performance.

(19)日本国特許庁（ＪＰ）　(12) **公開特許公報（A）**　(11)特許出願公開番号

特開平9－265409

(43)公開日　平成９年(1997)10月７日

(51)Int.Cl.[6]	識別記号	庁内整理番号	ＦＩ		技術表示箇所
Ｇ０６Ｆ　9/46	３６０		Ｇ０６Ｆ　9/46	３６０Ｂ	
13/00	３５７		13/00	３５７Ｚ	

審査請求　未請求　請求項の数３　ＯＬ　（全 38 頁）

(21)出願番号　特願平9－40713

(22)出願日　平成９年(1997)２月25日

(31)優先権主張番号　０８／６１３１０６

(32)優先日　1996年３月８日

(33)優先権主張国　米国（ＵＳ）

(71)出願人　390009531
インターナショナル・ビジネス・マシーンズ・コーポレイション
INTERNATIONAL BUSINESS MASCHINES CORPORATION
アメリカ合衆国10504、ニューヨーク州　アーモンク　（番地なし）

(72)発明者　モーハン・シャルマ
アメリカ合衆国78728　テキサス州オースチン　ルネッサンス・コート　14000　ナンバー1102

(74)代理人　弁理士　合田　潔　（外２名）

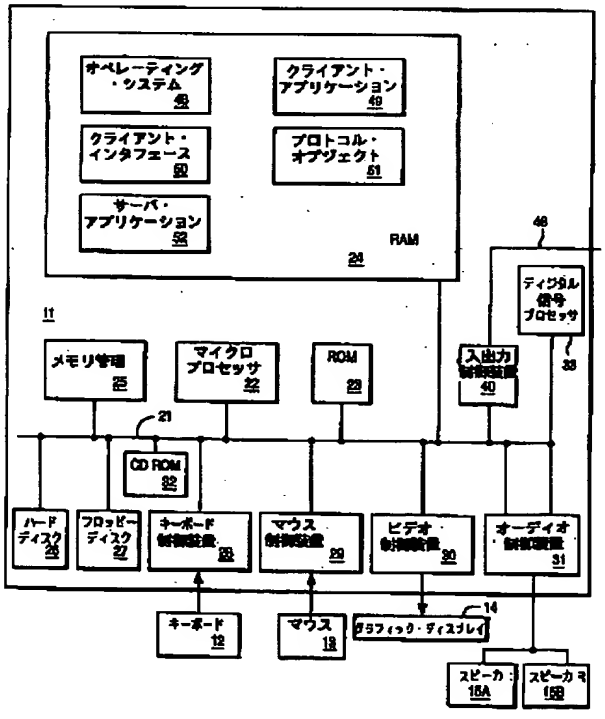
最終頁に続く

(54)【発明の名称】　高性能ユーザ・レベル・ネットワーク・プロトコル・サーバ・システムのための動的実行ユニット管理

(57)【要約】

【課題】　サーバ・システム内の実行ユニットのプールを動的に管理するための方法、システム、製品を提供する。

【解決手段】　そのプールは、クライアント・プロセスとサーバ・プロセスとの間の通信プロセス専用である。通信プロセス・プール内に最小数および最大数の実行ユニットが確立される。最小数の実行ユニットは、典型的なクライアント・ロードをサポートするのに必要な数である。最大数の実行ユニットは、サーバ・システムにとって過負荷にならずにピーク・クライアント・ロードをサポートするための上限である。クライアントによるサービス要求がサーバ・システムによって受信されると、いくつかの判定が行われる。まず、その要求に実行ユニットを割り当てると、通信プロセス・プール内の現行数の実行ユニットが最大数の実行ユニットを上回ることになるかどうかが判定される。上回る場合、クライアント要求は拒否される。



1

【特許請求の範囲】

【請求項１】サーバ・システム内の実行ユニットの、クライアント・プロセスとサーバ・プロセスとの間の通信プロセス専用のプールを動的に管理するための方法において、

通信プロセス・プール内に、典型的なクライアント・ロードをサポートするのに必要な数である最小数、およびサーバ・システムにとって過負荷にならずにピーク・クライアント・ロードをサポートするための上限である最大数の実行ユニットを割り振るステップと、

クライアントによるサービス要求をサーバ・システムが受信するステップとを含み、

受信した各クライアント要求ごとに、

受信したクライアント要求に実行ユニットを割り当てると、通信プロセス・プール内の現行数の実行ユニットが最大数の実行ユニットを上回ることになるかどうかを判定し、上回る場合にクライアント要求を拒否するステップと、

受信したクライアント要求に実行ユニットを割り当てると、その要求を行うクライアント・タスクに割り当てられた現行数の実行ユニットがクライアント・タスク用の割り当てられた実行ユニットの数を上回ることになるかどうかを判定し、上回る場合にクライアント要求を拒否するステップと、

判定ステップが否定であればクライアント要求を許可し、その結果、通信プロセス・プール内の実行ユニットがクライアント要求に割り当てられるステップとを含むことを特徴とする方法。

【請求項２】サーバ・システム内の実行ユニットの、クライアント・プロセスとサーバ・プロセスとの間の通信プロセス専用のプールを動的に管理するためのシステムにおいて、

通信プロセス・プール内に、典型的なクライアント・ロードをサポートするのに必要な数である最小数、およびサーバ・システムにとって過負荷にならずにピーク・クライアント・ロードをサポートするための上限である最大数の実行ユニットを割り振る手段と、

クライアントによるサービス要求をサーバ・システムが受信する手段と、

受信したクライアント要求に実行ユニットを割り当てると、通信プロセス・プール内の現行数の実行ユニットが最大数の実行ユニットを上回ることになるかどうかを判定する手段と、

受信したクライアント要求に実行ユニットを割り当てると、その要求を行うクライアント・タスクに割り当てられた現行数の実行ユニットがクライアント・タスク用の割り当てられた実行ユニットの数を上回ることになるかどうかを判定する手段と、

通信プロセス・プール内の実行ユニットをクライアント要求に割り当てることができると判定手段が確認した場合にクライアント要求を許可する手段とを含むことを特徴とするシステム。

2

【請求項３】サーバ・システム内の実行ユニットの、クライアント・プロセスとサーバ・プロセスとの間の通信プロセス専用のプールを動的に管理するためのコンピュータが読取り可能な媒体上のコンピュータ・プログラム製品において、

通信プロセス・プール内に、典型的なクライアント・ロードをサポートするのに必要な数である最小数、およびサーバ・システムにとって過負荷にならずにピーク・クライアント・ロードをサポートするための上限である最大数の実行ユニットを割り振るようにシステムに指示する手段と、

クライアントによるサービス要求をサーバ・システムが受信するようにシステムに指示する手段と、

受信したクライアント要求に実行ユニットを割り当てると、通信プロセス・プール内の現行数の実行ユニットが最大数の実行ユニットを上回ることになるかどうかを判定するようにシステムに指示する手段と、

受信したクライアント要求に実行ユニットを割り当てると、その要求を行うクライアント・タスクに割り当てられた現行数の実行ユニットがクライアント・タスク用の割り当てられた実行ユニットの数を上回ることになるかどうかを判定するようにシステムに指示する手段と、

通信プロセス・プール内の実行ユニットをクライアント要求に割り当てることができるという判定に応答してクライアント要求を許可するようにシステムに指示する手段とを含むことを特徴とするコンピュータ・プログラム製品。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】本発明は、一般に、コンピュータ・ネットワーク上の通信、ならびにこのような通信を容易にするコンピュータ・プロトコルに関する。より具体的には、本発明は、ネットワーク・プロトコル・アクセスのためのオブジェクト指向通信インタフェースに関する。

【０００２】

【従来の技術】非常に初期のコンピュータ・システムは、ディスプレイやプリンタなどの周辺装置と入力装置が接続された、スタンドアロン・プロセッサであった。各コンピュータ・システムは独立しており、コンピュータ・システム間の通信はほとんど行われていなかった。現在、ローカル・エリア・ネットワークや広域ネットワークなどのコンピュータ・ネットワークでは、複数のコンピュータ・システムを相互接続し、ネットワークに結合された各種コンピュータ・システムから得られるデータ、サービス、資源の共有を含む様々な利益を達成することが周知になっている。

【０００３】ネットワークにより各種コンピュータ・シ

ステム間で通信するために、多くの通信プロトコルが開発された。周知のネットワーク・プロトコルの例としては、システム・ネットワーク体系（ＳＮＡ）、伝送制御プロトコル／インターネット・プロトコル（ＴＣＰ／ＩＰ）、ネットワーク基本入出力システム（ＮｅｔＢＩＯＳ）、インターネット・パケット交換／順次パケット交換（ＩＰＸ／ＳＰＸ）などがある。その他の通信プロトコルも既知であり、広く使用され、ＩＳＯ、ＩＥＥＥ、その他の組織の様々な規格に記載されている。コンピュータ・ネットワークの理解を容易にするため、ネットワークの諸機能および関連ソフトウェアは一連の層として記述されることが多い。ネットワークにより配布されたアプリケーションの１つのコピーとその配布されたアプリケーションの別のコピーとの間のデータ転送は、基礎となる一連の通信層のサービスを使用することによって行われる。一般に、１つのコンピュータ・システム内の各層は、それぞれの層がそれぞれの対等層とやりとりするように、受信側コンピュータ・システム内に対応層を備えている。

【０００４】７層の開放型システム間相互接続（ＯＳＩ）モデルは、ネットワーク通信の説明のうち、最もよく知られたものの１つであるが、多くの通信実施態様では、１つまたは複数のＯＳＩ層を結合するかまたは省略している。ＯＳＩの物理層は、ネットワークと直接やりとりする最下層である。これは、物理的接続部を越えて実際にビット・ストリームをネットワークに伝送することを含む。第２の層は、物理層ストリームの多重化とフレーム化を行ってメッセージにするデータリンク層である。これは、エラー検出、同期情報、物理チャネル管理ももたらす。第３の層は、ネットワークによる情報の経路指定を制御するネットワーク層である。アドレス指定、ネットワークの初期設定、切替え、セグメント化、フォーマット化などのサービスはこの層に用意される。データ送達の肯定応答は、この層で行われる場合もあれば、データリンク層で行われる場合もある。

【０００５】第４の層は、透過データの送達、多重化、マッピングを制御するトランスポート層である。これより下の諸層で行われる最善の努力とは対照的に、アプリケーションが必要とすれば、信頼性の高い送達がこの層で行われる。欠落データの再伝送、順不同で送達されたデータの再配列、伝送エラーの訂正などのサービスは、通常、この層で行われる。第５の層は、トランスポート層からの情報を使用して、セッションと呼ばれるネットワーク内の２つのノード間の共通活動として個々のデータをグループ化するセッション層である。第６の層は、セッション層と第７の層であるアプリケーション層とのインタフェースを含むプレゼンテーション層である。プレゼンテーション層は、セッション層の整合性を損なわずに、アプリケーション層で使用するための情報を提供する。また、プレゼンテーション層はデータの解釈とフォーマットおよびコードの変換を行うのに対し、アプリケーション層はユーザ・アプリケーション・インタフェースと管理機能を提供する。

【０００６】もう１つの周知のネットワーク規格はＩＥＥＥ規格である。ＩＥＥＥモデルとＯＳＩとの主な相違点は、ＯＳＩの第２の層であるデータリンク層が媒体アクセス制御（ＭＡＣ）副層と論理リンク制御（ＬＣＣ）副層という２つの副層に分割されることである。媒体アクセス制御は、その制御アクセスにおける通信媒体への媒体アクセス接続を管理する。論理リンク制御は、関連するデータ・リンク制御が指定するプロトコルをサポートするための状態マシンを提供する。

【０００７】もう１つの周知の技術は、オブジェクトというプログラミング・エンティティにデータとメソッドをカプセル化するオブジェクト指向プログラミングである。公用インタフェースにより所与のメソッドとデータを保護することにより、オブジェクト指向プログラムは、各コンポーネントを他のコンポーネントに対する変更から隔離しながら、最小限の再プログラミングによって必要な諸機能を提供することができる。オブジェクト指向の技術、概念、規則に関する詳細な背景情報については、グレイディー（Grady Booch）によるObject Oriented Design WithApplications (The Benjamin/Cummins Publishing Company, 1990)およびB. マイヤー（Meyer）によるObject Oriented Software Construction (Prentice Hall, 1988)などの参考文献を参照されたい。

【０００８】マルチプロセッサ・ネットワークにおける通信プロトコルの一般領域に対してオブジェクト指向技術を応用しようという試みは、すでにいくつか行われてきた。以下に示すように、それは依然として本発明の実り多い分野である。

【０００９】

【発明が解決しようとする課題】サーバ・システム内の実行ユニットのプールを動的に管理するための方法、システム、製品であって、そのプールは、クライアント・プロセスとサーバ・プロセスとの間の通信プロセス専用である。通信プロセス・プール内に最小数および最大数の実行ユニットが確立される。最小数の実行ユニットは、典型的なクライアント・ロードをサポートするのに必要な数である。最大数の実行ユニットは、サーバ・システムにとって過負荷にならずにピーク・クライアント・ロードをサポートするための上限である。クライアントによるサービス要求がサーバ・システムによって受信されると、いくつかの判定が行われる。まず、その要求に実行ユニットを割り当てると、通信プロセス・プール内の現行数の実行ユニットが最大数の実行ユニットを上回ることになるかどうかが判定される。上回る場合、クライアント要求は拒否される。次に、その要求に実行ユニットを割り当てると、その要求を行うクライアント・タスクに割り当てられた実行ユニットの数がクライアン

ト・タスク用の割り当てられた実行ユニットの数を上回ることになるかどうかが判定される。上回る場合、クライアント要求は拒否される。判定結果が否定であればクライアント要求が許可され、それにより、通信プロセス・プール内の実行ユニットがクライアント要求に割り当てられる。

【００１０】

【課題を解決するための手段】システム・パフォーマンスを改善するために通信プール内の未使用の実行ユニットの数を増減する必要があるかどうかを判定するため、未使用の実行ユニットの数を定期的に検討する。

【００１１】上記の目的、特徴、および利点は、添付図面と以下の説明を参照すれば、さらに容易に理解されるだろう。

【００１２】

【発明の実施の形態】本発明は、複数通りのオペレーティング・システム下で様々なコンピュータまたは複数のコンピュータの集合で動作することができる。このコンピュータは、たとえば、パーソナル・コンピュータ、ミニ・コンピュータ、メインフレーム・コンピュータ、他のコンピュータからなる分散ネットワークで動作するコンピュータなどにすることができる。コンピュータの具体的な選択はディスクやディスク記憶装置の要件によってのみ制限されるが、ＩＢＭのＰＣシリーズのコンピュータであれば、本発明に使用できるはずである。ＩＢＭのＰＣシリーズのコンピュータの詳細については、IBM PC 300/700 Series Hardware Maintenance, Publication No. S83G-7789-03およびUser's Handbook IBM PC Series 300 and 700, Publication No. S83G-9822-00等を参照されたい。ＩＢＭのパーソナル・コンピュータが動作可能なオペレーティング・システムの１つは、ＩＢＭのOS/2 Warp 3.0である。ＩＢＭのOS/2 Warp 3.0オペレーティング・システムの詳細については、OS/2 Warp V3 Technical Library, Publication No. GBOF-7116-00等を参照されたい。

【００１３】代替実施例のコンピュータ・システムは、ＡＩＸ（TM）オペレーティング・システム上で動作するＩＢＭのＲＩＳＣシステム／６０００（TM）系列のコンピュータにすることもできる。ＲＩＳＣシステム／６０００の様々なモデルについては、RISC System/6000, 7073 and 7016 POWERstation and POWERserver Hardware Technicalという参考文献（資料番号SA23-2644-00）など、ＩＢＭの多くの資料に記載されている。ＡＩＸオペレーティング・システムについては、General Concepts and Procedure -- AIX for RISC System/6000（資料番号SC23-2202-02）ならびにＩＢＭのその他の資料に記載されている。

【００１４】図１には、システム・ユニット１１と、キーボード１２と、マウス１３と、ディスプレイ１４とを含むコンピュータ１０がブロック図形式で示されている。システム・ユニット１１は、１つのシステム・バスまたは複数のシステム・バス２１を含み、このシステム・バスには様々な構成要素が結合され、このシステム・バスにより様々な構成要素間の通信が行われる。マイクロプロセッサ２２は、システム・バス２１に接続され、同じくシステム・バス２１に接続された読取り専用メモリ（ＲＯＭ）２３とランダム・アクセス・メモリ（ＲＡＭ）２４によってサポートされている。ＩＢＭのＰＳ／２シリーズのコンピュータに搭載されているマイクロプロセッサは、３８６または４８６マイクロプロセッサを含む、インテルのマイクロプロセッサ・ファミリーの１つである。しかし、６８０００、６８０２０、６８０３０マイクロプロセッサなどのモトローラのマイクロプロセッサ・ファミリーや、ＩＢＭ製のＰｏｗｅｒＰＣチップまたはヒューレット・パッカード、サン、モトローラなどが製造するものなどの様々な縮小命令セット・コンピュータ（ＲＩＳＣ）マイクロプロセッサを含み、かつこれらに限定されないマイクロプロセッサも、特定のコンピュータで使用することができる。

【００１５】ＲＯＭ２３は、数あるコードのうち、やりとりや、ディスク・ドライブおよびキーボードなどの基本的なハードウェア操作を制御する、基本入出力システム（ＢＩＯＳ）を含んでいる。ＲＡＭ２４は、オペレーティング・システムとアプリケーション・プログラムのロード先になるメイン・メモリである。メモリ管理チップ２５は、システム・バス２１に接続され、ＲＡＭ２４とハード・ディスク・ドライブ２６およびフロッピー・ディスク・ドライブ２７とのデータの受渡しを含む直接メモリ・アクセス操作を制御する。同じくシステム・バス２１に結合されたＣＤ－ＲＯＭ３２は、マルチメディア・プログラムまたはプレゼンテーションなどの大量のデータを格納するために使用する。

【００１６】このシステム・バス２１には、様々な入出力制御装置、すなわち、キーボード制御装置２８、マウス制御装置２９、ビデオ制御装置３０、オーディオ制御装置３１も接続されている。予想通り、キーボード制御装置２８はキーボード１２用のハードウェア・インタフェースになり、マウス制御装置２９はマウス１３用のハードウェア・インタフェースになり、ビデオ制御装置３０はディスプレイ１４用のハードウェア・インタフェースであり、オーディオ制御装置３１はスピーカ１５用のハードウェア・インタフェースである。トークン・リング・アダプタなどの入出力制御装置４０により、ネットワーク４６を介して同様に構成された他のデータ処理システムへの通信が可能になる。

【００１７】本発明の好ましい実施態様の１つは、上記のように一般的に構成された１つまたは複数のコンピュータ・システムのランダム・アクセス・メモリ２４に常駐する複数の命令セット４８～５２である。コンピュータ・システムが要求するまで、命令セットは、ハード・

ディスク・ドライブ２６などの他のコンピュータ・メモリ、最終的にＣＤ－ＲＯＭ３２で使用するための光ディスクなどの取外し可能メモリ、最終的にフロッピー・ディスク・ドライブ２７で使用するためのフロッピー・ディスクに格納することができる。当業者であれば、この命令セットを物理的に格納すると、それが電気的、磁気的、または化学的に格納されている媒体が物理的に変化し、その媒体上にコンピュータが読取り可能な情報が乗ることを理解できるはずである。命令、記号、文字などによって本発明を記述すると便利であるが、これらの表現および同様の表現はいずれも適切な物理要素に関連付けられている必要があることに留意されたい。さらに、本発明は、比較や妥当性検査、あるいは人間の操作員に関連しうる他の表現によって記述されることも多い。ここに記載するいずれの操作でも、本発明の一部を形成するような人間の操作員によるアクションは一切不要であり、この操作は電気信号を処理して他の電気信号を生成するためのマシン操作である。

【００１８】このワークステーションが統合されているネットワークは、ローカル・エリア・ネットワーク（ＬＡＮ）または広域ネットワーク（ＷＡＮ）であり、後者は他のノードまたは既知のコンピュータ・アーキテクチャに基づいて動作するシステムのネットワークへのテレプロセシング接続を含む。いずれのノードでも、１つまたは複数の処理システムが存在可能であり、それぞれの処理システムはだいたい上記のように構成された単一ユーザ・システムまたは複数ユーザ・システムにすることができる。これらの処理システムは、サービスを要求するか供給するかに応じて、クライアント・ワークステーションまたはサーバ・ワークステーションとして動作する。特定の実施態様では、本発明は、様々な通信プロトコルのうちの１つまたは複数によって通信するネットワークによって相互接続された複数のＩＢＭ互換ワークステーション上で動作する。ソフトウェア・アプリケーションは、まとめてパッケージ化するか、個別のアプリケーションとして販売することができる。ローカル・エリア・ネットワークの簡単な説明は、ラリーＥ．ジョーダン（Larry E. Jordan）およびブルース・チャーチル（Bruce Churchill）著Communications and Networking For The IBM PC（Robert J. Brady発行、A Prentice Hall Company 1983）等に記載されている。

【００１９】好ましい実施例では、本発明は、オブジェクト指向プログラミング技法を使用してＣ＋＋プログラミング言語で実現される。Ｃ＋＋はコンパイル済みの言語である。プログラムは人間が読取り可能なスクリプトで作成され、このスクリプトがコンパイラという他のプログラムに供給され、マシンが読取り可能な数字コードが生成されるが、このコードはコンピュータにロードしてコンピュータが直接実行できるものである。Ｃ＋＋言語は、ソフトウェア開発者が他の開発者によって作成されたプログラムを容易に使用できるようにするような所与の特性を処理しながら、その破壊や不適正使用を防止するためにプログラムの再使用を大幅に制御する。Ｃ＋＋言語は周知のものなので、この言語について詳細に記載した論文やテキストは数多く存在する。

【００２０】当業者には既知のように、オブジェクト指向プログラミング技法は、「オブジェクト」の定義、作成、使用、破棄を含む。これらのオブジェクトは、データ要素とルーチン、すなわちデータ要素を操作するメソッドとを含む、ソフトウェア・エンティティである。データと関連メソッドは、ソフトウェアによってエンティティとして扱われ、そのように作成し、使用し、削除することができる。データと関数により、オブジェクトは、データ要素により提示可能なその属性と、そのメソッドにより表現可能なその挙動によって、実世界のエンティティをモデル化することができる。

【００２１】オブジェクトの定義は、オブジェクトそのものではないが、実際のオブジェクトを構築する方法をコンパイラに指示するテンプレートとして機能する「クラス」を作成することによって行う。たとえば、クラスは、データを操作する関数に含まれるデータ変数とステップの数とタイプを指定することができる。実際にはオブジェクトは、対応するクラス定義と、オブジェクト作成中に供給される引数などの追加情報とを使用してオブジェクトを構築する、コンストラクタという特殊な関数を使ってプログラム内に作成される。オブジェクトの破棄は、デストラクタという特殊な関数によって行われる。

【００２２】多くの利点は、オブジェクト指向プログラミング技法の３つの基本的特性、すなわち、カプセル化、多様性、継承から発生している。オブジェクトは、内部データ構造および内部関数の全部または一部を隠す、すなわち、カプセル化するために設計することができる。より具体的には、プログラム設計時にプログラム開発者は、データ変数の全部または一部と関連メソッドの全部または一部を「私用」すなわちそのオブジェクトのみが使用するものと見なしてオブジェクトを定義することができる。他のデータまたはメソッドは、「公用」すなわち他のプログラムが使用可能なものであると宣言することができる。他のプログラムによる私用変数およびメソッドへのアクセスは、そのオブジェクトの私用データにアクセスする公用メソッドを定義することによって制御することができる。この公用メソッドは、私用データと外部プログラムとのインタフェースを形成する。私用変数に直接アクセスするプログラム・コードを作成しようと試みると、コンパイラは、プログラムのコンパイル時にエラーを発生する。このエラーによって、コンパイル・プロセスが停止し、プログラムが実行できなくなる。

【００２３】多様性とは、全体的なフォーマットが同じ

であるが別々のデータで機能する複数のオブジェクトおよび関数が、別々に機能して一定の結果を生み出せるようにするものである。たとえば、加算メソッドを変数Ａプラス変数Ｂ（Ａ＋Ｂ）として定義することができる。ＡとＢが数字、文字、またはドルとセントであっても、これと同じフォーマットを使用することができる。しかし、この加算を実行する実際のプログラム・コードは、ＡとＢを構成する変数のタイプに応じて、大幅に異なってくる可能性がある。したがって、変数のタイプ（数字、文字、ドル）ごとに１つずつ、３通りのメソッド定義を作成することができる。メソッドの定義後、プログラムはその共通フォーマット（Ａ＋Ｂ）によってその加算メソッドをあとで参照することができ、コンパイル時にＣ＋＋コンパイラは、その変数タイプを検査することによって３つのメソッドのいずれを使用すべきかを判定する。その後、コンパイラは適切な関数コードを代入する。

【００２４】オブジェクト指向プログラミングの第３の特性は、プログラム開発者が既存のプログラムを再使用できるようにする継承である。継承により、ソフトウェア開発者は、クラスと、クラス階層によって関連付けられるようにそのクラスからあとで作成されるオブジェクトとを定義することができる。具体的には、クラスは、他の基本クラスのサブクラスと呼ぶこともできる。サブクラスは、その基本クラスの公用関数がサブクラスに含まれている場合のように、その公用関数のすべてを「継承」し、そのすべてにアクセスすることができる。また、サブクラスは、同じ形式の新しい関数を定義することによって、継承した関数の一部または全部を指定変更したり、修正することもできる。

【００２５】他のクラスの機能性を借用する新しいサブクラスを作成すると、ソフトウェア開発者は、その特定のニーズに合うように既存のコードを容易にカストマイズすることができる。

【００２６】オブジェクト指向プログラミングにより、他のプログラミングの概念が大幅に改善されるが、特に修正に使用可能な既存のソフトウェア・プログラムがまったくない場合には、依然としてプログラム開発には相当な時間と労力が必要になる。その結果、いずれも特定の環境で一般に検出されるタスクの実行向けの１組のオブジェクトと追加の雑ルーチンを作成するために、１組の事前定義相互接続済みクラスが用意されていることがある。このような事前定義クラスとライブラリは、通常、「フレームワーク」と呼ばれ、本質的に作業アプリケーション用の事前製作構造を提供する。

【００２７】たとえば、ユーザ・インタフェース用のフレームワークは、ウィンドウ、スクロール・バー、メニューなどを作成する１組の事前定義グラフィック・インタフェース・オブジェクトを提供し、これらのグラフィック・インタフェース・オブジェクト用のサポートと「デフォルト」挙動を提供する可能性がある。多くのフレームワークはオブジェクト指向技法に基づいているので、事前定義クラスを基本クラスとして使用することができ、組込みデフォルト挙動を開発者定義サブクラスが継承し、開発者がそのフレームワークを拡張し、特定の専門分野でカストマイズ済みの解決策を作成できるように修正または指定変更することができる。プログラマは元のプログラムを変更せず、むしろ元のプログラムの諸機能を拡張するので、このオブジェクト指向手法は従来のプログラミングより大幅に優れている。さらに、このフレームワークは体系的なガイダンスとモデル化を提供し、同時に、開発者は問題定義域に固有の特定のアクションを自由に供給できるようになる。

【００２８】以下に記載する本発明では、コンピュータ・ネットワーク内の様々なコンピュータ・システムのアプリケーション用のネットワーク・サービスを提供するためにネットワーキング・フレームワークを提供する。

【００２９】プロトコル・インタフェース・モデル

ここでは、オブジェクト指向のプロトコル・インタフェース・モデルおよび機構について説明する。このインタフェースの一般的な特徴により、ＯＳＩネットワーク・モデルのすべての層の開発が可能になる。したがって、すべてのネットワーク層は同様の構文になり、その意味は特定の層の仕様によって定義される。すなわち、ＯＳＩネットワーク・モデル内の各種の層を実現するオブジェクトは構文上は同様であるが、そのインプリメンテーションは特定の層の責任に応じて異なる可能性がある。さらに、ＴＣＰ／ＩＰなどの特定のプロトコル用のそれぞれの層を実現するオブジェクトは、ＮｅｔＢＩＯＳなどの別のプロトコルの同様の層とは、機能およびインプリメンテーションの点で異なる可能性がある。というのは、２つのプロトコルではそれぞれの層の責任が異なるからである。このモデルは、通信端点を定義し、端点を再使用し、ネットワーク事象を監視するための機構も提供する。これは、オブジェクト指向モデルなので、コードの再使用性、保守容易性、クライアント・アプリケーションに影響せずにインプリメンテーションを更新する能力など、オブジェクト指向インプリメンテーションの特徴をすべて継承している。

【００３０】１．通信端点の作成：通信端点の作成は、ネットワーク定義オブジェクトにより容易になっている。各ネットワーク定義オブジェクトは、クライアント・インタフェースの定義を含む。これは、クライアント・プログラムがアクセスを必要とすると思われる各種タイプのオブジェクトの抽象概念である。通信端点は、ネットワーク定義クラスから派生したTAccessDefinitionオブジェクトである。

【００３１】図２は、ネットワーク定義オブジェクトのクラス階層を示している。TNetworkDefinitionクラス１００は、端点をインスタンス化するためのメソッドInst

antiateDefinitionと、通信端点を破棄するためのメソッドDeinstantiationDefinitionをそれぞれ含んでいる。TNetworkDefinitionの新しいインスタンスを構築するためのメソッドまたは特定のクラス・オブジェクトを破棄するためのメソッドも用意されているが、これらのメソッドはすべてのクラスに共通なので、以下の説明では繰り返さないことにする。

【００３２】TAccessDefinitionオブジェクト１０１は、特定のプロトコル・タイプとその諸層を定義するプロトコル・インタフェース・オブジェクトを含んでいる。TAccessDefinitionオブジェクト１０１は、通信端点のハンドルとして機能する。さらに、TAccessDefinitionオブジェクト１０１は、その親であるTNetworkDefinitionから継承したメソッドに加え、すでに追加されているものの上にアクセス定義へのインタフェース層を加えるためのメソッドAddToTopと、一番下にすでに追加されているものにAccessDefinitionへのインタフェース層を加えるためのメソッドAddToBottomと、最も高いプロトコル層インタフェースを獲得するためのメソッドGetTopOfStackも含んでいる。TEventDefinitionクラス１０３は、ネットワーク事象を監視するために使用する。TEventDefinitionクラス１０３については、以下のNetworkEventの項でより詳細に説明する。

【００３３】２．ネットワーク・アドレス・クラス
TNetworkAddressクラス１１１は、すべてのプロトコル・アドレス・クラスとハードウェア・アドレス・クラスを定義するために使用する基本クラスである。図３は、プロトコル・アドレス・クラスのクラス階層を示している。TNetworkAddressクラス１１１は、アドレスのタイプをテストするためのメソッドIsOfAddressTypeと、グループ・アドレスかどうかをテストするためのメソッドIsGroupと、同報アドレスかどうかをテストするためのメソッドIsBroadCastと、マルチキャスト・アドレスかどうかをテストするためのメソッドIsMulticastと、ヌル・アドレスかどうかをテストするためのメソッドIsNullAddressとを含んでいる。このクラスは、アドレスのワイヤ長を獲得するためのメソッドGetWireLengthと、アドレスをヘッダにフォーマット化するためのメソッドAddressToWireと、ヘッダからアドレスを獲得するためのメソッドWireToAddressも含んでいる。ストリームが端点に着信するのか、それともストリームが端点から発信するのかを判定するための演算子も、このクラスの一部である。各クラスに固有のポインタを獲得するためのメソッドGetClassIdentifierは、このクラスによって提供される。

【００３４】TProtocolAddressクラス・オブジェクトのインスタンスは、プロトコル・アドレスを渡すことができるように、通信端点として機能する。TProtocolAddressオブジェクト１１３は、同報アドレスを獲得するためのメソッドBroadCastAddressと、ヌル・アドレスを獲得するためのメソッドNullAddressとを加える。THardwareAddressオブジェクト１１５のインスタンスは、必要に応じてアプリケーションにハードウェア・アドレスを渡すものである。同様に、THardwareAddressオブジェクト１１５は、同報アドレスを獲得するためのメソッドBroadCastAddressと、ヌル・アドレスを獲得するためのメソッドNullAddressとを加える。また、機能アドレスかどうかをテストするためのメソッドIsFunctionalも加える。TIEEE8023Addressクラスは、ＩＥＥＥ８０２．３というアドレス指定規則に応じてハードウェア・クラス・アドレスを変更したものである。さらに、これは、グループ・アドレスかどうかをテストするためのメソッドIsGroupと、グループ・アドレスに設定するためのメソッドSetGroupと、グループ・アドレスをリセットするためのメソッドClearGroupとを加える。他のメソッドとしては、同報アドレスを獲得するBroadCastAddress、マルチキャスト・アドレスかどうかをテストするIsMulticast、ヌル・アドレスを獲得するNullAddress、標準入力からアドレスを獲得するCanonicalAddressなどがある。

【００３５】TTCPAddr１１７とTNBAddr１１９は、それぞれＴＣＰ／ＩＰおよびＮｅｔＢＩＯＳのアドレス指定の詳細を表す具象アドレス・クラスの例である。同様に、TIEEE8023Addr１２１とTIEEE8025Addr１２３は、ＩＥＥＥ８０２．３およびＩＥＥＥ８０２．５のアドレス指定用の具象ハードウェア・アドレスを表している。

【００３６】３．プロトコル・インタフェース・クラス
本発明は、接続指向のトランザクションと無接続のトランザクションの両方をサポートする。また、本発明では、プロトコルとは無関係のアプリケーションが従う状態マシンについても説明する。プロトコル・インタフェース・クラスのオブジェクト階層を図４に示す。プロトコル・インタフェース・クラスは、プロトコルが備えていなければならないすべての共通関数用のメソッドを含む、MProtocolServe１３３という基本クラスから派生したものである。TProtocolInterfaceクラス１３５は、ネットワーク機能用の追加メソッドを含んでいる。これらのメソッドについては、以下の表１と表２に詳しく示す。ＴＣＰ／ＩＰなどのネットワーク・プロトコルは、そのTCPIPInterfaceクラスをTProtocolInterfaceから派生させて、デフォルト・インプリメンテーションを指定変更し、送信または受信要求上に有効フラグがあるかどうかという検査などのそれ専用の特定の特徴を追加する。TProtocolInterfaceクラスから派生する個別のクラスとしては、セッション層用のTSessionInterface１３７、トランスポート層用のTTransportInterface１３８、ネットワーク層用のTNetworkInterface１４１、ファミリー層用のTFamilyInterface１４３、データ・リンク層用のTDLCInterface１４５が用意されている。オブジェクト階層は図示の通りである。この実施例では、ＯＳＩネットワーク層が、ネットワーク層と、プロトコル

・ファミリー層という下位層とに分割される。ファミリー層は、経路指定情報など、ＯＳＩネットワーク層の非複製部分を含んでいる。ネットワーク層は、対等アドレスおよびローカルＳＡＰなど、特定の端点に関連する情報を含んでいる。これは、FamilyLayerオブジェクトなど、１つのオブジェクトだけがシステム内に常駐するようにして、すべてのグローバル情報とプロトコルとの関連を維持するために行われるものである。クライアント・アプリケーションによって各端点が作成され破棄されると、セッション層などの他のプロトコル層オブジェクトと、トランスポート層オブジェクトと、ネットワーク層オブジェクトは作成され破棄されるが、ファミリー層オブジェクトとデータリンク層オブジェクトは常駐したままになる。

【００３７】具象クラスは、個々のインタフェース・クラスから派生する。たとえば、以下に説明するように、プロトコル・スタックを構築するためにＴＣＰ／ＩＰなどの特定のプロトコル用のトランスポート、ネットワーク、ファミリー・インタフェース・クラスのために、具象クラスが用意されている。

【００３８】表１

前述のように、MProtocolServiceオブジェクトは、プロトコル層定義用の基本クラスとして機能する。MProtocolServiceオブジェクトに設けられているメソッドのリストを以下に示す。これらのメソッドの大部分は、純粋仮想関数である。

【００３９】

Bind	－ローカル・アドレスを初期設定しバインドする
Unbind	－ローカル・アドレスをアンバインドする
SendRelease	－正常解放開始
ReceiveRelease	－正常解放開始の肯定応答受信
GetLocalAddress	－ローカル・アドレスを獲得する
SetLocalAddress	－ローカル・アドレスを設定する
GetPeerAddress	－対等アドレスを獲得する
SetPeerAddress	－対等アドレスを設定する
GetProtocolInfo	－プロトコル情報を獲得する
SetProtocolInfo	－プロトコル情報を設定する
GetProtocolOptions	－プロトコル・オプションを設定する
GetRequestMode	－要求モードを獲得する
SetRequestMode	－要求モードを設定する
GetRetry	－プロトコル層再試行パラメータを獲得する
SetRetry	－プロトコル層再試行パラメータを設定する
GetTimeout	－プロトコル層タイムアウト・パラメータを獲得する
SetTimeout	－プロトコル層タイムアウト・パラメータを設定する
GetStatistics	－プロトコル層統計情報を獲得する
SetStatistics	－プロトコル層統計情報を設定する
IsSession	－プロトコル層がセッション層である場合に真を返す
IsTransport	－プロトコル層がトランスポート層である場合に真を返す
IsNetwork	－プロトコル層がネットワーク層である場合に真を返す
IsFamily	－プロトコル層がファミリー層である場合に真を返す
演算子<<=	－オブジェクトをデータ・ストリームに受信するための演算子
演算子>>=	－オブジェクトをデータ・ストリームに送信するための演算子

【００４０】表２

TProtocolInterfaceクラスに設けられた関数のリストを以下に示す。

【００４１】

15　　16

GetLayerIndex	－プロトコル層のインデックスを獲得する
CancelRequests	－現行未解決要求をすべて取り消す
ReceiveEvent	－このスタック上の受信事象
GetConnectionInfo	－接続情報およびメモリの制約を入手する
BorrowMemory	－ネットワーク・データ用にシステム・メモリを借用する
ReturnMemory	－ネットワーク・データ用のシステム・メモリを返す
GetAccessDefinition	－TAccessDefinitionへのポインタを獲得する
GetLocalAddress	－層のローカル・アドレスを獲得する
SetLocalAddress	－層のローカル・アドレスを設定する
GetPeerAddress	－層の対等アドレスを獲得する
SetPeerAddress	－層の対等アドレスを設定する
GetProtocolInfo	－層のプロトコル情報を獲得する
SetProtocolInfo	－層のプロトコル情報を設定する
GetProtocolOptions	－層のプロトコル・オプションを獲得する
SetProtocolOptions	－層のプロトコル・オプションを設定する
GetRequestMode	－層の要求モードを獲得する
SetRequestMode	－層の要求モードを設定する
GetRetry	－プロトコル層再試行パラメータを獲得する
SetRetry	－プロトコル層再試行パラメータを設定する
GetStatistics	－プロトコル層統計情報を獲得する
GetTimeout	－プロトコル層タイムアウト・パラメータを獲得する
SetTimeout	－プロトコル層タイムアウト・パラメータを設定する
Bind	－プロトコル・スタックをバインドする
Unbind	－プロトコル・スタックをアンバインドする
Receive	－ネットワーク・データを受信する
Send	－ネットワーク・データを送信する
Connect	－接続を確立しようという試みを開始する
ReceiveConnection	－接続を確立しようという試みを待つ
Disconnect	－接続を終了する
ReceiveDisconnect	－切断を待つ
AcceptConnection	－接続を確立しようという試みを受け入れる
RejectConnection	－接続を確立しようという試みを拒否する
Listen	－接続を確立しようという試みを聴取する
ListenForConnection	－接続を確立しようという試みを聴取する
SendRelease	－正常解放開始＝＞送信すべきデータはこれ以上ない
ReceiveRelease	－正常解放開始の肯定応答受信
演算子<<=	－オブジェクトをデータ・ストリームにストリーム・インするための演算子
演算子>>=	－オブジェクトをデータ・ストリームにストリーム・アウトするための演算子

【００４２】４．プロトコル層インプリメンテーション・クラス

前述のように、プロトコル・インタフェース・モデルは、プロトコル層にアクセスするためのオブジェクト・ベースのＡＰＩとして機能する。ＴＣＰ／ＩＰなどのプロトコル・スタックのインプリメンテーションを階層化した１組のリンク済みオブジェクトとして実現するには、TProtocolLayerクラスを使用する。図５は、プロトコル層クラスのクラス階層を示している。TProtocolLayerクラス１５１は、１つのプロトコル・インプリメンテ

ーションのすべての層の基本クラスとして機能する。

【００４３】TProtocolLayerクラス１５１は、各層で実施されるTransmit、Connect、Receive、Disconnectなどの関数用のメソッドを含んでいる。これらのメソッドについては、以下の表３に詳しく示す。

【００４４】ＴＣＰ／ＩＰなどのプロトコル用の具象クラスは、これらの層オブジェクトから派生し、その具体的なプロトコル層の意味を取り入れている。

【００４５】それぞれの子オブジェクト１５３～１６１は、これらのメソッドを継承し、特定のプロトコル層に該当する場合にはそのメソッドを指定変更する。

【００４６】表３

TProtocolLayerクラスの主な関数を以下に示す。

【００４７】

Dispatch	－インバウンド・パケットを上位層にディスパッチする
Transmit	－アウトバウンド・パケットを下位層に伝送する
DispatchEvent	－事象を上位層にディスパッチする
TransmitEvent	－事象を下位層に伝送する
ReceiveEvent	－事象の報告を可能にする
CancelReceiveEvent	－事象の報告を取り消す
InstantiateInterface	－層オブジェクトからインタフェース・オブジェクトを作成する
GetUpperProtocol	－次の上位層へのポインタを獲得する
GetLowerProtocol	－次の下位層へのポインタを獲得する
Bind	－プロトコル・スタックをバインドする
Unbind	－プロトコル・スタックをアンバインドする
Connect	－接続を確立しようという試みを開始する
ReceiveConnection	－接続を確立しようという試みを待つ
Disconnect	－接続を終了する
GetPacketQueue	－インバウンド・データ・パケットが得られるパケット待ち行列へのポインタを返す
ReceiveDisconnect	－切断を待つ
AcceptConnection	－接続開始を受け入れる
RejectConnection	－接続を確立しようという試みを拒否する
Listen	－接続を確立しようという試みを聴取する
ListenForConnection	－接続を確立しようという試みを聴取する
SendRelease	－正常解放開始＝＞送信すべきデータはこれ以上ない
ReceiveRelease	－正常解放開始の肯定応答受信
演算子<<=	－TProtocolLayerオブジェクトをデータ・ストリームにマーシャルするための演算子
演算子>>=	－TProtocolLayerオブジェクトをデータ・ストリームにアンマーシャルするための演算子

【００４８】５．プロトコル状態マシン

ここでは、接続指向操作と無接続操作の両方について可能なプロトコル・インタフェース・オブジェクトの状態について説明する。プロトコル状態マシンを図６および図７に示す。これらの状態マシンは、プロトコルで典型的な状態遷移を示しているが、プロトコルのインプリメンテーションによっては異なる選択も可能である。しかし、この状態マシンは、アプリケーションの可搬性を確実にし、プロトコルの独立性を可能にする。プロトコルの独立性を必要とするアプリケーションでは以下に説明する状態マシンを使用し、プロトコルの独立性をサポートするプロトコルによって状態マシンが実現されるはずである。

【００４９】特定のプロトコル・インタフェース層オブジェクト用のプロトコル端点は、以下に記載する状態のいずれかになるはずである。これらの状態は通常、アプリケーション状態である。層オブジェクトは、特定のプロトコルによって定義される状態マシンに従う。通常、これらの状態は、ユーザが各種の「端点」状態で行うことができる有効な呼出しと、アプリケーションの作成方法を示すためのものである。層状態は、層の意味と状態によって制御される。

【００５０】未初期設定状態２０１は、端点の最終状態でもある端点の開始状態を定義するものである。これは、アクセス定義が作成される前の状態である。

【００５１】TAccessDefinitionオブジェクトが作成されると（２０２）、端点は初期設定済みと呼ばれる。初期設定済み状態２０３では、Set操作を使用してインタフェース・オブジェクトを構築し初期設定できるはずである。Set操作はインタフェース・オブジェクトでローカルにキャッシュされるので、設定値の妥当性検査を行う余地はほとんどない。すなわち、InstantiateDefinitionメソッドによりプロトコル層オブジェクトが作成される前に、インタフェース・オブジェクト上で送信／受信容量を設定することができる。このような容量への制限に関する情報は層オブジェクトだけが把握しているので、このような値をインタフェース上で妥当性検査することは不可能である。というのは、AccessDefinitionオブジェクト上でInstantiateDefinitionメソッドが呼び出されるまで、層オブジェクトが存在しないからである。TAccessDefinitionオブジェクト２０４を破棄するよう要求すると、端点が初期設定済み状態２０３から未初期設定状態２０１に移行する。

【００５２】TAccessDefinitionオブジェクト上でインスタンス化要求２０６を行うと、端点が未結合状態２０７に移行する。未結合状態２０７は、層オブジェクトがインスタンス化された直後に発生する状態を定義する。値を修正するためにGet／Set操作を出すことができる。このような要求はもはやインタフェース・オブジェクトにキャッシュされないが、対応する層オブジェクトに送られて、処理され格納される。TAccessDefinitionオブジェクト上でインスタンス解除要求２０８を行うと、端点が初期設定済み状態２０３に移行する。

【００５３】ローカル・アドレスをバインドするためにBind操作２１０が出されると、端点が未結合状態２０７から結合状態２０９に移行する。無接続モードのデータ転送の端点は、「結合」状態になるとデータの送受信を開始することができる。結合状態２０９でスタックに対してアンバインド要求２１２が出されると、端点が未結合状態２０７に戻る。

【００５４】Listen要求２１４が出されると、端点が結合状態２０９から聴取状態２１３に移行する。プロトコル・スタックは、ユーザ指定の待ち行列サイズを使い尽くすまでそのローカル名に関する着信接続要求を受け入れる。着信接続２１６により、新しい活動プロトコル・スタックが作成される。

【００５５】活動端点でConnect要求２２０が出されると、端点が結合状態２０９から接続（クライアント）状態２１９に移行する。図７に示すように、無接続モードのサービスを使用する端点は、接続要求が正常に行われた後で「データ転送」状態２２５に入る。受動端点の場合、着信接続要求を受信すると、プロトコル・スタックは、層とインタフェース・オブジェクトのコピーと、受信した接続要求用の新しいTAccessDefinitionを作成する。新たに作成された端点は、その後、接続（サーバ）状態２２１になる。矢印は、聴取状態から接続（サーバ）状態まで点線上にある。アプリケーションは、接続を受け入れるか、または接続を拒否するかを選択することができる。AcceptConnectionでは、端点がデータ転送状態２２５になるはずである。接続要求２２６が拒否された場合、端点は非活動状態２２９に移行する。

【００５６】新たに作成したスタックが接続要求２２２を完了後、端点はデータ転送状態２２５に入る。ローカル・アプリケーションからDisconnect要求２２８を受信するか、または接続パートナーによって接続が終了された後に、接続端点が結合状態２０９に移行する。ただし、このような場合、アプリケーションはReceiveDisconnect要求を出して終了データを受信できることに留意されたい。

【００５７】着信接続要求がアプリケーションによって拒否されると、端点は非活動状態２２９に移行する。次に、TAccessDefinition破棄操作２３０を出すことによって、端点が破棄される。

【００５８】６．例

ここでは、ＴＣＰ／ＩＰなどのネットワーク・プロトコルに本発明のオブジェクト指向モデルを使用する方法の例を示す。ＴＣＰ／ＩＰインプリメンテーションは、トランスポート層と、ネットワーク層と、ＴＣＰ／ＩＰファミリー層とを含む。さらに、ネットワーキング・サブシステムは、システム内に常駐するTDatalink層オブジェクトを含み、ＴＣＰ／ＩＰのFamilyLayerオブジェクトはTDatalink層にアクセスする手段を備えている。ただし、この実施例では、TDataLink層オブジェクトはＴＣＰ／ＩＰ、ＳＮＡ、ＮｅｔＢＩＯＳなど、すべてのネットワーク・プロトコル・スタックに共通し、TDatalink層オブジェクトはTProtocolLayerクラス・オブジェクトから派生することに留意されたい。

【００５９】ＴＣＰ／ＩＰインタフェース（ＡＰＩ）クラスは、TProtocolInterfaceクラスから派生している。図８は、ＴＣＰ／ＩＰインプリメンテーションの一部のオブジェクトのクラス階層を示している。前述のように、TProtocolInterfaceとTProtocolLayerは、ＡＰＩおよびプロトコル・インプリメンテーション機能を提供するMProtocolServiceクラスの子クラスである。TTCPTINF２５１と、TTCPNINF２５３と、TTCPFINF２５５の各オブジェクト（まとめてTTCPXINFと呼ぶ）は、ＴＣＰ／ＩＰプロトコル用のTTransportInterfaceクラス、TNetworkInterfaceクラス、TFamilyInterfaceクラスを表すTProtocolInterfaceの具象クラスの例である。ただし、ＴＣＰ／ＩＰプロトコルには「セッション」という概念がないので、TSessionLayerクラスのインスタンスがないことに留意されたい。同様に、TTCPTIMP２５７と、TTCPNIMP

２５９と、TTCPFIMP２６１の各オブジェクト（まとめてTTCPXIMPと呼ぶ）は、ＴＣＰ／ＩＰプロトコル用のTTransportInterfaceクラス、TNetworkInterfaceクラス、TFamilyInterfaceクラスのインスタンスである。

【００６０】前述のように、TDatalinkLayer１６１は、この実施例のすべてのネットワーク・プロトコル用のインプリメンテーション・オブジェクトであり、TProtocolLayerクラス１５１のサブクラスである。

【００６１】また、ＩＰアドレスを渡すことができるように、アプリケーションにはTTCPProtocolAddressクラスが用意されている。TTCPProtocolAddressは、４バイトのＩＰアドレスと、２バイトのポート・アドレスとを含むことになる。

【００６２】ＴＣＰ／ＩＰプロトコルにアクセスするためにアプリケーションが必要とするプロセスについては、図９を参照して以下に説明する。ほとんど場合、「アプリケーション」は、ユーザ・レベル・アプリケーションがＡＰＩ呼出しを行うＢＳＤソケット・インタフェースなどの通信ＡＰＩ層になる可能性が最も高い。ソケット・インタフェースにより、アプリケーション開発者は、ネットワーク・プロトコル・オブジェクトの特定の構文を把握する必要がなくなるはずである。しかし、「アプリケーション」は、本発明のオブジェクト・モデルの名前に精通したオペレーティング・システム・サービスにもなりうる。以下の例では、アプリケーションがトランスポート層にアクセスする。アプリケーションはどの層にもアクセスすることができる。ネットワーク層に直接送話することも可能である。現在、ユーザ・アプリケーションがまったくないことが一般的である。これらのステップは、ＴＣＰ／ＩＰプロトコルの手続き上のインプリメンテーションのものと高レベルで非常によく似ている。しかし、それは、本発明のプロトコル・インタフェース・モデルを使用して、オブジェクト指向のやり方で実現されている。

【００６３】ステップ３０１では、ＴＣＰ／ＩＰプロトコルにアクセスするために通信端点が作成される。これは、まずTAccessDefinitionオブジェクトを作成することによって行われる。たとえば、Ｃ＋＋を使用して「新しいTAccessDefinition」が構築される。次に、TAccessDefinition内のメソッドを使用して、TTCPxINFインタフェース・オブジェクトが作成され、AccessDefinitionに追加される。その後、TAccessDefinitionオブジェクト上でInstantiateProtocolメソッドが呼び出され、次にそれによりTTCPxIMP層オブジェクトが作成される。

【００６４】ステップ３０３では、ステップ３０１で作成された通信端点にアドレスが結合される。これは、用意されたTTCPProtocolAddressクラス・オブジェクトから必要なＩＰアドレスを備えたTTCPProtocolAddressオブジェクトを作成することによって行われる。次に、そのアドレスをバインドするためにTTCPTINF->Bind()メソッドが呼び出される。このステップは、バインドの意味を含むプロトコル・インプリメンテーション層上でTTCPIMP->Bind()メソッドを起動することになる。

【００６５】次に、ステップ３０５では、（TTCPTINFオブジェクトで）TTCPTINF->Connect()メソッドを呼び出して接続要求を開始することによって、アプリケーションが聴取対等機能に接続する。これにより、（TTCPTIMPオブジェクトで）TTCPTIMP->Connect()メソッドが起動され、次にこのメソッドは、下位層、すなわち、TTCPNIMP->Connect()メソッドとTTCPFIMP->Connectメソッド用のTTCPNIMPオブジェクトとTTCPFIMPオブジェクトをそれぞれ呼び出すことにより、ＴＣＰ／ＩＰ接続をセットアップするための必要なステップを実行する。

【００６６】正常接続後、ステップ３０７では、結果のプロトコル・スタックによりデータを送受信することができる。アプリケーションは、TTCPTINF->Send()メソッドとTTCPTINF->Receive()メソッドを呼び出して、ネットワーク・データを送受信する。TTCPTINF->Send()は次にTTCPTIMP->Xmit()メソッドを呼び出して、ＴＣＰプロトコルのデータ伝送を開始する。データは、各プロトコル層のXmit()関数を使用してプロトコル層オブジェクトからプロトコル層オブジェクトに渡され、通信アダプタによりTDataLinkLayerオブジェクトに送達される。同様に、受信機能の場合は、TDataLinkLayerが物理層からデータを受信し、それを適切なプロトコル・ファミリー層に渡し、次にその層が適切なスタックに渡す。しかし、一実施態様では、クライアントからのデータ受信要求が行われるまでデータが待ち行列化され、データはデータ待ち行列からクライアントにコピーされる。

【００６７】ステップ３０９では、所望の通信の完了後に接続が閉じられる。アプリケーションは、TTCPTINF->Disconnect()メソッドを呼び出して、接続終了を開始する。これにより次にTTCPTIMP->Disconnect()が呼び出され、このメソッドは特定のインプリメンテーションの場合にファミリー層までそのメソッドを送信する可能性のあるTTCPTIMPのＴＣＰ／ＩＰ切断状態マシンを処理する。

【００６８】最後に、ステップ３１１では、TAccessDefinitionオブジェクトを削除することによって、端点が閉じられる。

【００６９】図１０を参照すると、様々なオブジェクトとネットワーク層との関係を理解するのに役立つだろう。同図の左側には、図８および図９に関連して前述したＴＣＰ／ＩＰ実施例が示されている。ＴＣＰ／ＩＰアプリケーション３１３、３１５は、いずれもアプリケーション層にあるソケットＡＰＩ３１７を介して、以下の層のオブジェクト指向プロトコル・スタックに連絡する。通信端点を作成するためにソケットＡＰＩが使用するTProtocolAddressオブジェクトは図示していない。それぞれのＴＣＰ／ＩＰアプリケーション３１３、３１５

ごとに個別の通信端点、すなわち、個別のTProtocolAddressオブジェクトが必要である。

【００７０】前述のように、ＴＣＰ／ＩＰにはセッション層という概念がないので、ソケットＡＰＩ３１７はトランスポート層内のTTCPTINFオブジェクト２５１によってやりとりする。前述のように、TAccessDefinitionオブジェクト３１８は、トランスポート層、ネットワーク層、ファミリー層のTTCPxINFオブジェクト２５１、２５３、２５５を含む。ユーザ・レベル・プロセスがファミリー層のネットワークによりプロトコル・スタックに連絡することは比較的まれなことであるが、主にオペレーティング・システム・サービスにより、これらの層に連絡するためにTTCPNINFオブジェクト２５３とTTCPFINFオブジェクト２５５が提供される。

【００７１】TTCPTINFオブジェクト２５１は、TTCPTINF-->Connect()メソッドによりTTCPTIMPオブジェクト２５７に接続する。これにより、TTCPNIMPオブジェクト２５９に接続するためのTTCPTIMP-->Connect()メソッドと、TTCPFIMPオブジェクト２６１に接続するためのTTCPNIMP-->Connect()メソッドが起動される。また、TDataLinkLayerオブジェクト１６１に接続するTTCPFIMP-->Connect()メソッドが起動される。同図に示す通り、TTCPTIMP２５７、TTCPNIMP２５９、TTCPFIMP２６１、TDatalinkLayer１６１の各オブジェクトは、それぞれトランスポート層、ネットワーク層、ファミリー層、データリンク層にある。前述のように、送受信メソッドにより、物理層内の物理アダプタ３１９を介してプロトコル・スタック上でネットワーク・データを送信することができる。

【００７２】好ましい実施例では、通信端点が作成され削除されたときに、TDataLinkLayerオブジェクト１６１とTTCPFIMPオブジェクト２６１は不変であり、単独である。それぞれの活動端点ごとに、残りのオブジェクト（２５１、２５３、２５５、２５７、２５９、３１８）のそれぞれの個別のスレッドとインスタンスが必要になる。当業者であれば容易に分かるように、何千ものクライアント接続が行われる可能性のあるネットワーク・プロトコル・サーバの場合、このような配置に関連するオーバヘッドによってパフォーマンス上の問題が発生する可能性がある。以下の動的実行ユニット管理の項で説明するように、本発明は、パフォーマンスを最大にするために実行スレッドを動的に管理するための手段を提供する。

【００７３】同図の右側には、ＮｅｔＢＩＯＳのインプリメンテーションが示されている。したがって、本発明は、システムが複数のネットワーク・プロトコルを実行する複数のアプリケーションをサポートするマルチプロトコル環境を企図するものである。図１０に示すように、ＮｅｔＢＩＯＳアプリケーション３３１、３３３は、ソケットＡＰＩ３１７または特殊ＮｅｔＢＩＯＳＡＰＩ層３３５のいずれかを介して下位層とやりとりすることができる。したがって、本発明は、本発明のオブジェクト指向プロトコル・スタックにアクセス可能な複数のアプリケーションも企図するものである。このようなアプリケーションは、オブジェクト・ベースまたは手続きベースのユーザ・レベル・アプリケーションまたはオペレーティング・システム・レベル・プロセスにすることができる。

【００７４】前述のように、同様のプロセスを使用してＮｅｔＢＩＯＳスタックが確立される。ただし、ＮｅｔＢＩＯＳにはセッション層という意味がないので、ＡＰＩ層３１７、３３５はセッション層のTNSINFオブジェクト３３７に接続することに留意されたい。ＮｅｔＢＩＯＳインタフェース・オブジェクトであるTNSINF３３７、TNTINF３３９、TNNINF３４１、TNFINF３４３と、ＮｅｔＢＩＯＳプロトコル層オブジェクトであるTNSIMP３４７、TNTIMP３４９、TNNIMP３５１、TNFIMP３５３が作成された後、TNSINFオブジェクト３３７によりプロトコル層オブジェクトを介してTDataLinkLayerオブジェクト１６１まで通信経路が確立される。ＴＣＰ／ＩＰ実施例のように、TDataLinkLayerオブジェクト１６１とTNFIMPオブジェクト３５３は不変かつ単独であり、それぞれの活動通信端点ごとに残りのオブジェクトのそれぞれの個別のインスタンスが作成され削除される。

【００７５】本発明では、TDataLinkLayer１６１に結合された複数のアダプタ３１９、３２１に対処することができる。上位層は通信端点までの適切な経路指定を行うので、異なるネットワーク・プロトコルに対して同時にこれらのアダプタを使用することができる。さらに、これらのアダプタは、イーサネットとトークン・リングなど、異なるタイプのものにすることができるので、サーバは各種のネットワークからの要求に対応することができる。

【００７６】７．プロトコル・ユーティリティ・クラス
このクラスは、TProtocolInterfaceクラスとTProtocolLayerクラスが使用するものである。このクラスのほとんどは、インタフェース・クラスとインプリメンテーション・クラスの様々なメソッドのパラメータとして機能する。このクラスは、提案したモデルの一般的な特徴を強調するものである。

【００７７】ここでは、プロトコル・インタフェースが使用する重要なクラスの一部について説明する。

【００７８】ａ．TProtocolInfoクラス
このクラスは、サービス・タイプ、最大メッセージ・サイズ、優先データ長など、そのプロトコルの様々な特性を識別するものである。このクラスが提供する重要なメソッドの一部を以下に示す。

【００７９】これらは、端点の作成時に端点の特性を指定するために使用する。

【００８０】

GetPSDUSize －プロトコル・サービス・データ・ユニットの最大サイズを獲得する

SetPSDUSize －プロトコル・サービス・データ・ユニットの最大サイズを設定する

GetEPSDUSize －優先プロトコル・サービス・データ・ユニットの最大サイズを獲得する

PSDUSize －優先プロトコル・サービス・データ・ユニットの最大サイズを設定する

GetConnectionDataSize －接続データの最大サイズを獲得する

SetConnectDataSize －接続データの最大サイズを設定する

GetDisconnectDataSize －切断データの最大サイズを獲得する

SetDisconnectDataSize －切断データの最大サイズを設定する

GetDisconnectDataSize －切断データの最大サイズを獲得する

SetDisconnectDataSize －切断データの最大サイズを設定する

GetServiceType －接続／無接続などのサービス・タイプを獲得する

SetServiceType －サービス・タイプを設定する

SetServiceType －サービス・タイプを設定する

SetServiceFlags －サービス・フラグを設定する

【００８１】ｂ．TProtocolOptionsクラス

このクラスは、サービス品質を含む、プロトコル固有のオプションを定義するために使用する。しかし、この基本クラスは、特定のサービス品質を含んでいない。具象クラスがTProtocolOptionsから派生し、その固有のオプションを含むものと思われる。

【００８２】

GetLingerTime －プロトコル・リンガ時間を獲得する

SetLingerTime －プロトコル・リンガ時間を設定する

GetSendbufSize －プロトコル送信バッファ・サイズを獲得する

SetSendbufSize －プロトコル送信バッファ・サイズを設定する

GetRcvbufSize －プロトコル受信バッファ・サイズを獲得する

SetRcvbufSize －プロトコル受信バッファ・サイズを設定する

【００８３】ｃ．TSendModifiersクラス

TSendModifiersクラスは、TProtocolInterface::SendメソッドとTProtocolLayer::Xmit()メソッドにおけるネットワーク送信機能に関連するフラグを修飾するものである。送信タイムアウトのサポートの他に、送信機能を左右しうる指示は以下の通りである。

【００８４】

kPush －即時処理を要求する

kEdnOfMessage －メッセージの終わりをマークする

kExpeditedData －データを優先データとして扱う

kNotify －クライアント・バッファが使用可能なときに送信側に通知する

kSendAll －「ｎ」バイトがすべてバッファされるまでブロックする

kNoFragmentation －データを断片化しない

【００８５】このクラスでサポートされる重要な関数の一部を以下に示す。

【００８６】

GetSendModifier －SendModifier用の値を獲得する

SetSendModifier －SendModifierをＯＮに設定する

ClearSendModifier －SendModifierをＯＦＦに設定する

GetTimeout －送信タイムアウトを獲得する

SetTimeout －送信タイムアウトを設定する

【００８７】ｄ．TReceiveModifiersクラス

このクラスは、TProtocolInterface::Receive()関数内の受信フラグを定義するために使用する。

【００８８】このクラスは、ネットワーク・データを受

信しながら、フラグとタイムアウトを設定するためのメソッドと定義を含む。以下の通りである。

kPeek	－ピーク・ユーザ・データ
kExpeditedData	－優先データを受信する
kReceiveAll	－「ｎ」バイトが受信されるまでブロックする
kBroadcastData	－同報データグラムを受信する
kDiscardBufferOverflow	－受信バッファからオーバフローするメッセージの一部を破棄する
kDatagramAny	－通常データグラムまたは同報データグラムのいずれかを受信する

【００９０】このクラスの重要な関数の一部を以下に示す。

【００９１】

GetReceiveModifier	－ReceiveModifier用の値を獲得する
SetReceiveModifier	－ReceiveModifierをＯＮに設定する
ClearReceiveModifier	－ReceiveModifierをＯＦＦに設定する
GetTimeout	－受信タイムアウトを獲得する
SetTimeout	－受信タイムアウトを設定する

【００９２】ｅ．SendCompletionクラス

ネットワーク・データ送信操作に応答してSendCompletionオブジェクトが返される。これは、送信機能の状況を示し、送信バイトの総数も示す。状況指示は以下の通りである。

【００９３】

kNormal	－正常完了
kTimeout	－送信タイムアウト
kCancelled	－アプリケーションが送信要求を取り消した

【００９４】このクラスの重要なメソッドの一部を以下に示す。

【００９５】

GetSTatus	－SendCompletion状況を獲得する
SetStatus	－SendCompletion状況を設定する
GetBytesTransferred	－送信するクライアント・データのバイト数を獲得する
SetBytesTransferred	－送信するクライアント・データのバイト数を設定する

【００９６】ｆ．TReceiveCompletionクラス

ネットワーク・データ受信要求に応答してReceiveCompletionオブジェクトが返される。これは、受信機能の状況と、データ受信バイトの総数を示す。状況指示は以下の通りである。

【００９７】

kNormal	－正常完了
kPushed	－送信側がデータを「プッシュした」
kNoMoreData	－ストリームが終了するかまたは受信パイプが閉じられた
kEndOfMessage	－メッセージの終わりが検出された
kExpeditedData	－優先データを受信した
kTimeout	－受信タイムアウト
kMessageTruncated	－部分メッセージであり、残りは破棄される
kCancelled	－取消し要求が処理された
kMore	－追加データの受信準備ができている

【００９８】このクラスの重要な関数の一部を以下に示す。

【００９９】

GetStatus	－ReceiveCompletion状況を獲得する
SetStatus	－ReceiveCompletion状況を設定する
GetBytesTransferred	－受信するクライアント・データのバイト数

を獲得する
SetBytesTransferred　　－受信するクライアント・データのバイト数
を設定する

【０１００】ネットワーク事象管理
ここでは、複数のネットワーク・プロトコル・スタック上で複数のネットワーク事象にアクセスし、それを様々なプロトコル層で格納するという問題に対するオブジェクト指向の解決策を開示する。本発明では、ＯＳＩネットワーク・プロトコル層モデルに基づくすべてのネットワーク事象の単一かつ一貫した１組のクラス定義を提供する。各総称ＯＳＩ層ごとに事象が定義されているので、これらの層の各インプリメンテーションは追加のサブクラスを定義することによって、追加の層固有の事象を加えることを選択できるはずである。この解決策は、オブジェクト指向モデルなので、ＯＯ技術で多様性と継承を完全に利用する。

【０１０１】ＯＳＩ層に基づく事象の分類により、プロトコル層で格納され報告される事象が明確にカテゴリー化される。このカテゴリー化により、アプリケーションにとって重要であれば、特定の層について事象を取り出すことができる。このような事象は、データが受信済みで受信に使用可能である、接続が打ち切られたなど、確立されているクライアントの通信セッションの条件を記述することができる。クライアントは、単一の呼出しを使用して、複数のプロトコル・スタックのいずれのプロトコル層上でもこれらの非同期事象を監視することができる。本発明は、アプリケーションが個々の端点事象を探す必要がなくなるように、ＴＣＰ／ＩＰやＮｅｔＢＩＯＳなどの複数のプロトコルによる事象報告に及ぶ。さらに、ネットワーク事象機構により、層オブジェクト間でネットワーク事象を送受信するためにネットワーク・プロトコル層間でのやりとりが容易になる。

【０１０２】１．事象の説明
事象は、TProtocolEventという基本クラスを使用して定義される。図１１は、様々な事象オブジェクト間のクラス階層を示している。図１１は図２に似ているが、TEventDefinitionクラス１０３のサブクラスであるTProtocolEventクラス４０１も示している。TProtocolEventクラス４０１は、列挙されたクラスの事象と、すべてのＯＳＩ層で一般に使用される１組の事象と、個々のプロトコル層事象用の値の予約範囲を含む。TProtocolEventクラスは、事象とメンバーシップの質問を設定し獲得するためのメソッドを有する。一実施例のネットワーク事象はビット・ベクトルとして設定されている。このクラスでは、GetCntメソッドがこのオブジェクト内に現在ある事象の数を返す。これは、この場合のオプション事象のリストを指定する典型的な方法であり、一般に使用するもう１つの方法は、ビット・マスクを使用する方法である。オブジェクト内にすべてのデータを隠すというオブジェクト指向技術の能力のために、単一オブジェクトだけが必要であり、そのオブジェクトがそれらを獲得するためのメソッドを提供する。

【０１０３】特定のネットワーク・プロトコルによっては、TProtocolEventクラスで事象値を再定義することによって、プロトコル事象を定義することができる。TProtocolEventの重要なメソッドの一部は、識別されたインデックスの事象を返すGetEventと、リスト内の特定の事象（メンバーシップ）を探索するHasAと、オブジェクト内の事象の数を返すGetCntと、端点を示すアクセス定義を返すGetAccessDefinitionと、端点用のアクセス定義を設定するSetAccessDefinitionと、TProtocolEventオブジェクト内の単一事象を設定するSetEventである。

【０１０４】２．ＯＳＩ層インプリメンテーションへの事象インタフェース
上記のプロトコル・インタフェース・モデルの項で述べたように、好ましい実施例ではネットワーク・プロトコル層が一連のTProtocolLayerオブジェクトとして実現され、それぞれが特定のＯＳＩ層を表している。これらのオブジェクトは互いに連鎖され、プロトコル・スタックを完成させている。たとえば、ＴＣＰ／ＩＰスタックは、トランスポート層、ネットワーク層、データリンク層用のオブジェクトを備えることができる。各プロトコル層オブジェクトは２つの事象インタフェース、すなわち、TProtocolLayerオブジェクトに定義されたReceiveEventインタフェースとCancelEventインタフェースとを有する。これらの層では、上位層のオブジェクトに事象を渡すためにDispatchEventメソッドを使用し、下位プロトコル層に事象を送信するためにTransmitEventメソッドを使用する。適切な層に事象を格納することは、プロトコル・スタックのインプリメンテーションの責任である。

【０１０５】TProtocolLayerクラスの事象関連メソッドは、上位層に事象をディスパッチするDispatchEventと、下位層に事象要求待ち行列を伝送するTransmitEventと、EventRequestQueueにTProtocolEventオブジェクトを追加することにより事象（複数も可）を報告するReceiveEventと、EventRequestQueueを無視するCancelReceiveEventである。

【０１０６】３．アプリケーションによる事象監視
それぞれのネットワーク・プロトコル層にオブジェクト・ベースのＡＰＩ用の基本クラスを提供するTProtocolInterfaceクラス・オブジェクトは、ネットワーク事象を受信するために端点が使用できる機能を含んでいる。TProtocolInterfaceクラスの事象関連機能は、特定の通信端点用にTProtocolEventオブジェクトに設定された事象（複数も可）を受信するReceiveEventメソッドである。

【０１０７】アプリケーションが複数の通信端点での事

象の監視を必要とするときは、TEventDefinitionオブジェクトを使用する。TEventDefinitionオブジェクトは、事象を監視するためのオブジェクト端点として機能する。このオブジェクトのインスタンスは、アプリケーションの要求に応じて作成され削除される。TEventDefinitionオブジェクトの主なメソッドは、TEventDefinitionオブジェクトのインスタンスをインスタンス化するInstantiateDefinitionと、TEventDefinitionオブジェクトのインスタンスを破棄するDeInstantiateDefinitionと、TProtocolEventオブジェクトのアレイまたは待ち行列で複数のスタックからの事象を受信するReceiveEventAnyと、保留事象要求を取り消すCancelRequestである。

【０１０８】ａ．１つの通信端点での事象の受信
アプリケーションは、特定のネットワーク層用のTProtocolInterfaceクラスのインスタンスでReceiveEventメソッドを使用して、特定の端点で事象を受信することができる。ただし、端点はそのプロトコル用のTProtocolInterfaceオブジェクトを使用してプロトコルにアクセスすることに留意されたい。アプリケーション・プログラムは、必要な事象をTProtocolEventオブジェクトに設定し、次にReceiveEventメソッドを呼び出す。TProtocolInterfaceオブジェクトは、事象が発生していればその事象を含むTProtocolEventオブジェクトを返す。

【０１０９】１つの通信端点で事象を受信するプロセスを図１２の流れ図に示す。この例では、トランスポート層が一番上の層であり、事象はネットワーク層に格納され、アプリケーションからの要求はトランスポート層に対して行われる。

【０１１０】もう１つの代替方法は、事象を受信するためにその層への直接の事象インタフェースをセットアップする方法である。しかし、このプロセスでは、あまり有利ではない事象についてもすべての層をそのように処置しなければならなくなる。１つの点から獲得する方が良好なので、クライアントは選択基準として事象ベクトルを使用してすべての層から事象を獲得するように呼出しを行う。

【０１１１】図１２は、単一端点での事象受信機構を示している。端点で１つまたは多数の事象を受信するための要求は、TProtocolInterface::ReceiveEvent()メソッド４１１を使用してインタフェース・オブジェクトに対して行われる。結果的に端点から報告されるTProtocolEventsを含むProtocolEventQueue４１３が内部で作成される。次に、TProtocolLayer::ReceiveEvent()メソッド４１５を使用して対応する層オブジェクトにその要求が送られる。同図は、層オブジェクト間でどのように要求が送られるかを示している。TProtocolEventQueueはReceiveEvent要求内のパラメータとして送られる。プロトコル層オブジェクトは、事象が発生するたびに、TProtocolEventQueueを無効にするTProtocolLayer::CancelEvent()要求を受信する前に、この待ち行列にTProtocolEventを待ち行列化する。要求された事象に応じて、TxxTransportLayerは、TProtocolLayerクラスのTransmitEvent()メソッドを使用してTxxNetworkLayerに要求を送ることができる（４１７）。同様に、事象要求が下位層に到達する必要がある場合には、TxxNetworkLayerは、要求をTxxFamilyLayerに送ることもできる（４１９）。

【０１１２】事象が非同期に発生する場合、TProtocolLayerオブジェクトは、DispatchEvent()を使用してその事象を上位層に報告することができる（４２１）。次にネットワーク層は、この事象をTxxTransportLayerに報告する（４２３）。TxxTransportLayerは報告された事象をProtocolEventQueueに待ち行列化する（４１５）。次に、事象は呼出し側に報告される（４１１）。

【０１１３】ｂ．複数の通信端点での事象の受信
アプリケーションは、TEventDefinitionオブジェクトを使用することにより、複数の通信端点の事象のための端点を作成することができる。図１１に示すように、TEventDefinitionクラス１０３はTNetworkDefinitionクラス１００から派生している。TEventDefinitionオブジェクト１０３は、任意のプロトコルで任意の数の端点の事象を監視するための端点として機能する。アプリケーション・プログラムにとって関心のある各通信端点ごとに、必要な数のProtocolEventsがTProtocolEventオブジェクトに設定される。すなわち、その事象が監視対象となるすべての端点について、２タプル（端点、プロトコル事象）の対が形成される。その後、アプリケーションは、様々な端点から要求された事象の組を渡して、TEventDefinitionオブジェクトのReceiveEventAny()メソッドへの呼出しを行う。ReceiveEventAnyは、TProtocolEventsの待ち行列に事象を入れて復帰するか、またはタイムアウトする。ただし、すべての通信端点はTAccessDefinitionクラスを使用して作成されることに留意されたい。

【０１１４】図１３は、複数の端点での事象受信の流れを示している。

【０１１５】図１３は、複数の端点での事象受信機構を示している。アプリケーションは、事象端点として機能するTEventDefinitionオブジェクトを作成する（４５１）。次にアプリケーションは、端点当たり１つずつ、TProtocolEventオブジェクトのアレイを作成する（４５３）。端点での要求された事象のアレイがパラメータとしてTEventDefinition::ReceiveEventAny()に渡される（４５５）。ReceiveEventAny()メソッドは、TProtocolEventQueueを作成し（４５７）、事象が探索されるすべての端点に待ち行列を送る（４６１）。ReceiveEvent要求を介して要求を受信する（４６３）TxxProtocolLayerは、それより下の層に事象要求を伝送することができる。これは、どの端点についても同じである（４６５）。TxxProtocolLayerは、事象を非同期に受信すると、上位層にその事象をディスパッチするか、またはTProtocolEventQueueで事象を報告することができる。事

象がTProtocolEventQueueに入ると、すべての関連端点のTxxProtocolLayerにTxxProtocolLayer::CancelEvent()が送られる。次に、ReceiveEventAnyメソッド４５５は、TProtocolEventQueueからすべてのTProtocolEventsを取り出し（４５９）、それを呼出し側に報告する。報告する事象がまったくない場合には、待ち行列が空であるはずである。

【０１１６】いずれかの端点から応答が受信された後、すべての端点はCancelEvent要求を受け取り、ProtocolEventQueueオブジェクトの寿命が終了したことを示す。その要求が取り消された場合でも、事象は依然としてプロトコル層に保管される。クライアントは、次に新しい事象を獲得するときに、別のReceiveEventとともに戻ることができる。この実施例では、クライアントからのプル・モデルを使用するが、クライアントにとって便利なときにクライアントが事象を獲得することを意味する。呼出し側には収集された１組のProtocolEventsが返される。

【０１１７】動的実行ユニット管理

マルチタスキング・システムでは、ＴＣＰ／ＩＰなどのネットワーク・プロトコル・スタックは、クライアント／サーバ・モデルを使用して効率よく実現することができる。サーバは、多数のクライアントをサポートするためにマルチタスキング技法を使用してユーザ・レベル・タスクにすることができる。多数のクライアントをサポートする際に十分なパフォーマンスを保証するため、固定した１組のサーバ資源を特定のプロセスに事前割振りすることが必要な場合も多い。重要なサーバ資源の１つは、割り振られたサーバ実行ユニットの数である。ＯＳ／２やWindows NTなどの最新のオペレーティング・システムでは、実行の基本ユニットはスレッドである。現在の多くのシステムは「軽量」スレッド（高速スレッド切替えを意味する）を備えているが、残念ながら、スレッド作成／削除は、何千ものクライアントをサポートするために拡大したときに本発明が提案するようなネットワーク・プロトコル・サーバのパフォーマンス要件をサポートできるほど十分「軽量」ではない。

【０１１８】さらに、スレッドの数が増すにつれて、システム、すなわち、サーバのパフォーマンスは大幅に低下する。スレッド切替えのオーバヘッドは、システム内に割り振られたフットプリントに加え、スレッドの数が増すにつれて比例して増加する傾向がある。これは、ネットワーク・セッション下の全二重通信で２つのスレッドに入れてデータの送受信を実行するなど、同じクライアントからの複数の同時要求をサポートする必要がありそうなネットワーク・サーバの場合にさらに複雑になる。その結果、複数の同時クライアント対応専用のサーバ内のスレッドの数は非常に大きくなる可能性がある。したがって、スレッドなどのサーバ実行ユニットを効率よく管理することは、十分なパフォーマンスを備えた多数のクライアントをサポートするサーバの能力にとって重要なことである。

【０１１９】ここでは、ネットワーク・プロトコル・サーバが実行スレッドの数を動的に管理するための方法を開示する。この方法は、多数のクライアントに対応するのに必要なスレッド資源を低減する。個々のクライアントがサーバ割振りのスレッド資源をすべて消費してしまわないように、進入制御が設けられている。同時に、サーバ・スレッド資源を待って特定のクライアントをいつまでも停止させておくことはない。

【０１２０】１．サーバ実行ユニット管理

サーバ・スレッドは、サーバ・スレッド・プールで作成され管理される。まずサーバが始動すると、サーバ管理スレッドが作成される。好ましいオブジェクト指向の実施例では、これは、以下に記載するクライアント・ポート・プールとネットワーク・サーバ・スレッド・プールを管理するためのメソッドを含むTNetworkServerクラス・オブジェクトをインスタンス化することによって行われる。サーバ管理スレッドは、スレッド・プールでのサーバ・スレッドの作成または削除を調整することにより、サーバ・スレッド・プールの管理を担当する。サーバ管理スレッドは通常、休眠状態である。これは、要求信号とタイマによって定期的に起こされる。

【０１２１】すべてのクライアント要求サービスは、最初にサーバとのセッションを確立しなければならない。セッション要求は、session_portという単一のサーバ通信端点に送られる。サーバとの通信セッションが確立した後、各クライアントにはclient_portという固有の通信端点が割り当てられる。好ましい実施例では、TAccessDefinitionオブジェクトとサーバ側で割り当てられた各client_portとの間に１対１の対応が存在する。

【０１２２】クライアントとサーバは、ＲＰＣ機構を使用して互いにやりとりすることができる。遠隔プロシージャ呼出し（ＲＰＣ）機構は、後述するように必要な要求経路指定と進入制御を行う。ＲＰＣ機構は、基礎となるオペレーティング・システムの内部プロセス通信の内部プロシージャ呼出し（ＩＰＣ）サービスの上に構築することができる。サーバ・タスクの位置の特定に加え、基本ＲＰＣプリミティブは以下の通りである。

【０１２３】クライアント側

○　SendRequest (server_port(セッションまたはクライアント), request_data, reply_data)

サーバ側

○　ReceiveRequest (request_data)

○　SendReply (reply_data)

【０１２４】すべてのクライアントのclient_portはclient_portプールに収集される。その割当てclient_portを使用して、クライアントはサーバにネットワーク要求を出すことができる。サーバは、client_portプールからの要求を管理し、スレッド・プール内のスレッドを任

意の数のclient_portプールから受け取ったサービス要求に割り当てる。図１４は、オブジェクト指向ネットワークの第１の実施例でのclient_portプールとスレッド・プールの制御及び管理の流れを示しているが、クライアントおよびサーバ・アプリケーション・オブジェクトは対称マルチプロセッサ（ＳＭＰ）などのマルチプロセッサ構成の各種プロセッサに割り振られたメモリ内に常駐し、両者はオブジェクト指向ＲＰＣインタフェースを介してやりとりする。

【０１２５】複数のクライアント・マシン５００は、ネットワーク５０５によりサーバ・マシン５０３に連絡する。それぞれのクライアント・マシン５００は、それぞれがクライアント要求を行うことができる複数のクライアント・アプリケーション・オブジェクト５０７を常駐させておくことができる。図示の通り、クライアント・オブジェクト５０７は、ＲＰＣ／ＡＰＩ層５０９の適切なＲＰＣ／ＡＰＩオブジェクトに自分の要求を送り、その層は異なるアドレス空間で動作するＩＰＣオブジェクト５１１を使用してサーバへの接続を行う。プロトコル・スタックは、ネットワーク５０５への通信経路を確立する。この実施例によれば、クライアント・マシンまたはクライアント・タスクが要求されたサーバでその許容量の通信スレッドを超えているかどうかを判定するために、以下に詳述するスレッド・カウント・オブジェクトがプロトコル・スタックに追加されている。超えていない場合、通信経路は作成されず、クライアント・プロセスは使用可能なスレッドを待つように指示されるか、またはプロトコル・スタックが単にクライアント・プロセスへのサービスを拒否することになる。同図にはサーバが１つだけ示されているが、スレッド・カウント・オブジェクトは、ネットワーク内の複数のサーバ・マシンでクライアント口座を追跡することができる。

【０１２６】サーバへの通信経路が確立されると想定して、プロトコル・スタック５１３は、サーバ側の通信端点であるセッション・ポートを割り当てるセッション・ポート・オブジェクト５１９に送るクライアント要求を受信する。クライアント要求には、クライアント・プール・オブジェクト５２１からクライアント・ポートが割り当てられる。クライアント・ポートは、サーバ側のスレッド・プール５２２に使用可能な十分なスレッドがある場合にクライアント・プロトコル・スタックに返されるクライアント通信端点である。セッション・ポート割振りと、クライアント・ポート・プールと、スレッド・プールとを管理する１組のネットワーク・プロトコル・サーバ・オブジェクトについては、以下に詳述する。スレッドと、クライアント・ポートと、セッション・ポートが割り振られると、サーバ・セットのＲＰＣ／ＡＰＩオブジェクト５１５と、上方向のサーバ・サービス５１７への通信を続行することができる。

【０１２７】動的実行ユニット管理は、手続きベースのシステムでも実行することができる。また、図１５に示すように、この機構は、メモリがクライアント空間５３１とサーバ空間５３３に分割される単一プロセッサ・マシンや、複数のクライアント・タスク５３５、５３７、５３９がサーバ・タスク５４０と同じマシンで動作する環境に応用することもできる。クライアント・タスクは、その要求をクライアント側のＲＰＣライブラリ・インタフェース５４１を介してサーバ・タスクに送る。要求はまず、サーバ側のＲＰＣライブラリ５４３によってサーバ側で処理される。次に、セッション・ポート・プロセス５４５によってセッション・ポートが割り振られ、クライアント・ポート・プール５４７からクライアント・ポートが割り振られる。プロトコル・スタック５５１がネットワーク５５３へのクライアント要求を処理できるように、スレッド・プール５４９からスレッドが割り当てられる。

【０１２８】プール内のスレッドの数は、複数のシステム構成変数によって制御される。これらの変数の修正は、ユーザ構成プログラムを使ってシステム管理者が行うことができる。

【０１２９】○（MinThreads）－これは、クライアント要求に対応するために予約されたスレッドの最小数である。この限界は、典型的なサーバ・ロードのクライアントをサポートするために使用するスレッドの最小数を維持するためのものである。

【０１３０】○（MaxThreads）－これは、クライアント要求に対応するために予約されたスレッドの最大数である。この限界は、サーバ・システムの過負荷にならずにピーク・ロードのクライアントの対応専用となるスレッド資源に関する上限として使用するものである。

【０１３１】○（MaxReq）－これは、所与のクライアントに割り振られる同時要求の最大数である。

【０１３２】○（TotalThreads）－これは、すべてのクライアント要求に対応するために作成されたスレッドの総数である。この値は、必ずMinThreadsとMaxThreadsとの間になる。

【０１３３】○（ReservedThreads）－これは、すべてのクライアント・セッションをサポートするために予約されたスレッドの総数である。

【０１３４】○（Unusedthreads）－これは、プール内の未使用スレッドの数である。

【０１３５】○（Clientthreads）－これは、各クライアントごとに、サーバが対応する活動要求の数である。

【０１３６】プール内のスレッドの数は、同時に処理される活動クライアント要求の数に基づいて、動的に増加または減少する。

【０１３７】スレッドの数を管理するために使用するプロセス、すなわち、オブジェクト指向実施例のメソッドについては、図１６～１９を参照して以下に詳述する。

【０１３８】図１６を参照すると、ステップ５７５のサ

ーバ初期設定では、＜MinThreads＞個のスレッドが作成され、スレッド・プールに挿入される。また、このステップでは、（UnusedThreads）と（TotalThreads）が（MinThreads）に設定され、（ReservedThreads）が０に設定される。ステップ５７７では、新しいクライアントがサーバとのセッションを要求する。サーバ側では、クライアント要求を受信後に、（ReservedThreads）＋（MaxReq）＜＝（MinThreads）であるかどうかを判定するためにステップ５７９でテストが行われる。この結果が真であれば、既存のクライアントと新しいクライアントをサポートするために十分なスレッドが存在するので、スレッド・プールを調整するためのアクションは一切行われない。

【０１３９】ステップ５８１では、サーバがクライアント・タスクにclient_portを割り当てて返す。ステップ５８３では、（ReservedThreads）が（MaxReq）だけ増分され、新しいクライアントがセッションを要求するまで待つためにステップ５７７に戻る。

【０１４０】ステップ５７９のテストが否定の場合、クライアント要求が許可された場合に最大数のスレッドを超えるかどうかを判定するためにステップ５８５でテストが行われる。（MinThreads）＜（ReservedThreads）＋（MaxReq）＜（MaxThreads）かどうかのテストが行われる。この結果が真であれば、スレッド限界を超えておらず、プロセスはステップ５８１に移行し、クライアント・タスクにクライアント・ポートを割り当てて返す。ステップ５８３では、（ReservedThreads）が（MaxReq）だけ増分される。後述するように、この図のプロセスはこの分岐と上記の分岐の場合に同じように見えるが、管理スレッドは、新しいカウント（ReservedThreads）に基づいて次に起こされたときにスレッド・プール内のスレッドを非同期に増加する。

【０１４１】ステップ５８５のテストが肯定の場合、すなわち、（ReservedThreads）＋（MaxReq）＞（MaxThreads）である場合、ステップ５８７でクライアント要求が拒否される。したがって、サーバは、過負荷の状況を回避するためにクライアント・セッション進入制御を管理する。ステップ５８９では、スレッドが他のクライアント・タスクから解放されるまで、サーバは待ち状態に入り、おそらくクライアントに待ち状態を通知するはずである。あるいは、サーバは、クライアント要求を拒否し、後でクライアントがもう一度試行できるようにするだけである。

【０１４２】以下の擬似コードは、このプロセスの実現例を示している。

【０１４３】

```
New client request session with server
On the server -
  If (ReservedThreads) + (MaxReq) <= (MinThreads),
     (ReservedThreads) increment by (MaxReq)
    Assign and return a client_port to the client
  If (MinThreads < (ReservedThreads) + (MaxReq) < (MaxThreads)
    Increment (ReservedThreads) by (MaxReq)
    Assign and return a client_port to the client
  If (ReservedThreads) + (MaxReq) > (MaxThreads)
    Reject the network request
```

【０１４４】図１７には、クライアント要求の進入制御を管理するためのクライアント側プロセスが示されている。ステップ６０１では、クライアント・タスクがサーバに割り当てられたクライアント・ポートを使用して新しいネットワーク要求を出す。ステップ６０３では、クライアント・タスクがサーバ側の許容数のスレッドのうち使用可能な複数のスレッドを備えているかどうかを判定するためにテストが行われる。これを判定するために、（ClientThreads）＜（MaxReq）かどうかのテストを使用することができる。備えている場合、ステップ６０７でクライアント要求はサーバに送られる。ステップ６０９では、現行クライアント・スレッドの数が増分される。

【０１４５】サーバ側の許容数のスレッドのすべてが使用中である場合、ステップ６０５でネットワーク要求が拒否されるか、またはこのクライアントからの同時要求の数が（MaxReq）より少なくなるまで待機状態に入る。待機状態になっているときに、許容スレッドの１つが解放された場合、プロセスはステップ６０５からステップ６０７に進むことができる。

【０１４６】図１７には、クライアント要求に対応するためにスレッド・プール内のサーバ・スレッドを割り当て、プール内のスレッドの数を調整するためのサーバ側のプロセスも示されている。ステップ６１１では、クライアント要求の処理に必要な期間中、サーバ・スレッドがクライアント要求に割り当てられる。ステップ６１３では、（UnusedThreads）変数が減分される。ステップ６１５では、スレッド・プール内のスレッドの数が許容できるかどうかを判定するためにテストが行われる。（（UnusedThreads）＜（MinThreads）＆（TotalThreads）＜（MaxThreads））かどうかのテストは、このステップを実行するためのテストの１つである。スレッド・プール内の使用可能なスレッドの数が許容できない場合には、スレッド・プール内のスレッドの数を増加するよ

うに管理スレッドに通知される。プロセスはステップ６１９で終了する。

【０１４７】以下の擬似コードは、このプロセスの実現例を示している。

```
Client issues network request to the server
  If (ClientThreads) < (MaxReq),
    Send the request to the server.
    Increment (ClientThreads)
  else
    Reject the network request
```

【０１４８】

```
Assign server thread to service client request
Decrement (UnusedThreads)
If (UnusedThread)<(MinThread)&(TotalThread)<(MaxThread)
    Increase threads in the thread pool
```

【０１４９】

【０１５０】図１８には、サーバがクライアント・ネットワーク要求を完了したときにクライアントとサーバが従うプロセスが示されている。サーバ側では、ステップ６２５でクライアント要求にどのような処理が必要であってもその処理をサーバが完了している。ステップ６２７では、プロトコル・スタックを介してクライアント要求への応答が送られる。次に、ステップ６２９では、サーバ・スレッドがスレッド・プールに返される。次に、ステップ６３１では、（UnusedThreads）変数が増分される。

【０１５１】クライアント側では、ステップ６３５でプロトコル・スタックがサーバからの応答を受け取る。ステップ６３７では、他のクライアント・タスクが通信スレッドのクライアント割振り分を使用できるようにするために、（ClientThreads）変数が減分される。ステップ６３９では、（ClientThread）が（MaxReq）限界を上回っているためにサーバに要求を送るために待機している同じクライアントから任意のスレッドを起こすための信号が出される。ステップ６４１では、プロセスが図１７のステップ６０７に移行する。

【０１５２】図１９には、サーバ管理スレッド・プロセスが示されている。サーバ管理スレッドは、スレッド・プール内の未使用スレッドの数がある程度低い限界より下がったときにスレッド割振りのためにタイマまたは信号によって起こされる。ステップ６５３では、さらにスレッドを追加する必要があるかどうかを判定するためのテストが行われる。適当なテストの１つは、（（UnusedThreads）＜（MinThreads）＆（TotalThreads）＜（MaxThreads））かどうかである。そうではない場合、ステップ６５４で管理スレッドが休眠状態に戻る。そうである場合、ステップ６５５で（（UnusedThreads）－（MinThreads））個のスレッドをプールに加えるという式に応じて、通信スレッドがスレッド・プールに追加される。ステップ６５７では、（UnusedThreads）が（MinThreads）と等しくなるように設定される。ただし、UnusedThreadsが（MinThreads）限界より下がったときだけ、ただちにスレッドが追加されることに留意されたい。それ以外の場合は、次のタイマ間隔までスレッドが遅延される。ステップ６５９では、（TotalThreads）変数が追加したスレッドの数だけ増分される。管理スレッドはステップ６５４の休眠状態に戻る。

【０１５３】図１９では、パフォーマンスを改善するために通信スレッド・プール内の未使用スレッドの数を低減するためのサーバ管理スレッド方法も示している。ステップ６６１では、通信スレッドの割振りまたは割振り解除のために定期的にスレッドを起こすタイマによって、スレッドが起こされる。ステップ６６３では、スレッド・プール内のスレッドが多すぎるかどうかを判定するためのテスト、たとえば、（（ReservedThreads）＜（MinThreads））＆（（UnusedThread）＞（MinThreads））かどうかなどが行われる。そうではない場合、ステップ６６４で次の間隔までスレッドが休眠する。そうである場合、ステップ６６５でプール内のスレッドの数が１だけ低減される。ステップ６６７では、（TotalThreads）が１だけ減分される。所定の期間中にまったく活動が発生しない場合、結果的に（TotalThreads）変数が（MinThread）に戻ることになる。ステップ６６９では、スレッド・プール内の未使用スレッドが少なすぎるかどうかをテストが判定する。（ReservedThreads）＞（TotalThreads）＆（（ReservedThreads）＜（MaxThreads））かどうかの式を使用することができる。そうである場合、（（ReservedThreads）－（TotalThreads））を加えるという式に応じて、スレッドがスレッド・プールに追加される。ステップ６７３では、（TotalThreads）変数が（ReservedThreads）に設定される。管理スレッドはステップ６７４の休眠状態に戻る。

【０１５４】本発明の利点としては、セッションおよび通常クライアント要求の進入制御がある。さらに、クライアント要求に対応するためのスレッド資源がクライアント・タスクに認められた限界まで必ず保証されている。したがって、サーバ・スレッド資源を待ってクライアントをいつまでも停止させておくことはない。

【０１５５】すべてのクライアント間で共用可能な事前割振りスレッドの場合、クライアント要求に対応するスレッド実行経路でスレッドの作成または削除が行われないので、サーバのパフォーマンスによってクライアント要求への応答時間が改善される。非活動の期間中に割振り済みスレッドの数を（MinThreads）まで定期的にプルーニング・バックすることによって、オーバコミット・スレッド資源が最小限になる。プール内のサーバ・スレッドの総数は、構成値（MaxThreads）まで拡大することができる。非活動スレッドは最小限に維持されるので、これによりシステム・オーバヘッドが低減される。

【０１５６】（MaxThreads）と（MinThreads）は構成限界であり、システム管理者がアクセスできるようにすることができるので、本発明によって動的スレッド資源調整が達成される。これらは、手作業で構成して、導入システムに最適な最小値に調整することによって調整することができる。あるいは、クライアントと同時クライアント要求の数について、１日の様々な時間に統計をとり続け、これらの統計に基づいて自動的に（MinThreads）と（MaxThreads）の値を計算し調整することもできる。

【０１５７】マルチスレッド化クライアント／サーバ・システムではスレッド・プールの概念が一般に使用されてきたが、クライアントの数とクライアントの使用法に基づく通信スレッド・プールの動的調整はこれまで行われていない。本発明は、長期または短期で動作するクライアント要求に対応するために同様に適用可能である。さらに、ユーザ・レベルで動作するネットワーク・プロトコル・サーバを実現するためのスレッド・プールの使い方は、先行技術では実施されていない。

【０１５８】本発明は、サーバがマルチタスキング化され、多数のクライアントを効率よくサポートしなければならない、いかなるクライアント／サーバ・システムでも実施することができる。この解決策は、ＩＢＭメインフレームＭＶＳや多くのＵＮＩＸベースのシステムなど、軽量の実行ユニットを持たないシステムには特に有用である。

【０１５９】前述のように、上記のアルゴリズムを使用するネットワーク・プロトコル・サーバは、１組のネットワーク・サーバ・クラスを定義することによって、オブジェクト指向技術を使用して実現することができる。このようなクラスの定義例を以下に示す。

【０１６０】

```
クラスTNetworkServer：公用TNetworkThreadHandle
{
  公用：
        TNetworkServer－クラス・コンストラクタ
        ～TNetworkServer－クラス・デストラクタ
        AddClientPort－クライアント・ポート・プールにポートを追加する
        RemoveClientPort－クライアント・ポート・プールからポートを除去す
る
        AddServerThread－ThreadPoolにスレッドを追加する
        DeleteServerThread－ThreadPoolからスレッドを削除する
        ExecuteMgmtThread－管理スレッド入口
        ExecuteThread－NetworkThread入口点
  私用：
        RegisterNames－クライアントにとって使用可能になるべきサーバ名を
公表する
```

【０１６１】

```
ネットワーク・サーバ実行用に使用するネットワーク・スレッド・クラス
クラスTNetworkThreadHandle：
{
  公用：
        TNetworkThreadHandle－クラス・コンストラクタ
        TNetworkThreadHandle－クラス・デストラクタ
        Fork－クラス関数を実行する新しいスレッドを開始する
        Join－スレッドが完了するまで待つ
        Release－スレッドがそのオブジェクトを削除できることを示す
```

【０１６２】

```
ネットワーク・クライアント／サーバ通信用のＲＰＣクラス
class TNWRPCMessage {
公用:
```

43 44

```
    virtual TNWRPCMessage();
    virtual ～TNWRPCMessage();
    /* クライアント側メソッド: SendRequest() */
    virtual kern_return_t SendRequest (const port_t&
clientport,
        void* buffer, int& buflen);  // buffer used for
inbound/outbound
    virtual kern_return_t SendRequest (const port_t&
sessionport,
        port_t& clientport, void* buffer, int& buflen);
    /* サーバ側メソッド: ReceiveRequest() &
Reply&Receive() */
    virtual kern_return_t ReceiveRequest (const port_t&
sessionport
        void* buffer, int& buflen);
    virtual kern_return_t ReceiveRequest (const port_t&
clientport_pool,
        void* buffer, int& buflen, port_t& clientport);
    virtual kern_return_t SendReply (const port_t&
newclientport,
        void* buffer, int& buflen, port_t& sessionport);
    virtual kern_return_t SendReply (const port_t&
clientport,
        void* buffer, int& buflen);
};
```

【０１６３】クライアント／サーバ環境でのネットワーク要求の表現

ここでは、クライアント／サーバ・モデル用のネットワーク・プロトコル要求のオブジェクト指向表現を示す。プロトコルＡＰＩがオブジェクト指向で、プロトコル・インプリメンテーションもオブジェクト指向の場合には、プロトコル要求のオブジェクト指向表現は重要なものになる。ネットワーク・プロトコルにアクセスするためのオブジェクト・ベースのクライアント要求がサーバに送られ、このサーバがこれらの要求を適切なネットワーク・プロトコル層オブジェクトに送達する。本発明は、クライアント・ネットワーク要求を転送し、その要求をサーバ上に常駐する適切なネットワーク・プロトコル層に送達し、要求の結果をクライアント・アプリケーションに取り出すための新しい方式を提示する。クライアント要求は「ネットワーク操作」オブジェクトで折り返されるが、そのオブジェクトはサーバがネットワーク・プロトコル層オブジェクトに要求を提示できるように必要な情報をすべて含んでいる。

【０１６４】以下のシナリオを検討されたい。クライアントＡＰＩは１組のオブジェクト指向ネットワーク・プロトコル・インタフェース要求を含み、プロトコルはオブジェクトのスタックとして実現され、それぞれのオブジェクトは特定のＯＳＩ層を表している。ネットワーク・プロトコル・スタックは、ネットワーク・サーバ上に常駐する様々なネットワーク・プロトコルを提供し、クライアントとサーバとの間でやりとりするためにＲＰＣまたはＩＰＣなどの通信機構が存在する。クライアントが何らかのデータの送信のようなネットワーク活動を要求した場合、サーバ上の適切なネットワーク・プロトコル・スタックにその要求を伝送する必要がある。本発明は、このようなクライアント要求をサーバに転送するための統一方式を提示する。この方式は多様性を利用しているので、このような要求を発送し、それをサーバで処理するプロセスは、どの要求およびプロトコル・スタックについても変わらない。

【０１６５】ネットワーク・プロトコルにアクセスするためのクライアント・インタフェースは、主にTProtocolInterfaceクラスによる。このプロトコル・インプリメンテーションは、TProtocolLayerクラスに基づいてそれぞれのＯＳＩ層を表している。このネットワーク・サブシステムは、ＴＣＰ／ＩＰやＳＮＡなどの各プロトコル用のTFamilyLayerオブジェクトと、TDataLinkLayerオブジェクトから構成され、どちらのオブジェクトもシステム内に常駐している。セッション層、トランスポート層、ネットワーク層を表すためのTProtocolLayerオブジェクトのスタックはすべてのクライアント端点ごとに作成され、クライアント通信端点はTAccessDefinitionオブジェクトによって記述される。このような概念、関連クラス、その機能については、いずれも上記のプロトコ

ル・インタフェース・モデルの項で説明する。

【０１６６】１．ネットワーク操作オブジェクト

クライアントからサーバへのネットワーク要求とその逆のネットワーク要求は、いずれもTNetworkOperationオブジェクトを使用して転送される。TNetworkOperationオブジェクトは、サーバ内のプロトコル層にクライアント・ネットワーク要求を伝送し、その結果をサーバからクライアントにリレーするための機構を提供する。

【０１６７】TNetworkOperationクラスは、すべての要求の基本クラスである。クライアント・インタフェースがサーバに対して行う各要求ごとに、TNetworkProtocolからサブクラスが派生している。TProtocolInterfaceメソッドのそれぞれについて、対応する「操作オブジェクト」クラスが１つずつ定義されている。図２０は、いくつかのクライアント要求用のクラス・オブジェクトのクラス階層を示している。TNetworkOperationクラス７０１は階層の一番上に位置する。TConnectOpクラス７０３と、TSendOpクラス７０５と、TReceiveOpクラス７０７と、TDisconnectOpクラス７０９と、TBindOpクラス７１３は、いずれも基本クラスから派生したもので、TProtocolInterfaceクラスの接続操作、送信操作、受信操作、切断操作、バインド操作にそれぞれ対応する。GetPeerNameおよびGetLocalNameなどの獲得および設定要求は、TGetSetNetworkOpクラス７１１という１つの操作クラスにバンドルされている。

【０１６８】TNetworkOperationオブジェクトは、ネットワーク・サーバでの対応を必要とする要求を満足するためにTProtocolInterfaceによって作成される。TNetworkOperationから派生したクラスは、インタフェースからの特定の要求を表す。TNetworkOperationオブジェクトは、要求が適切なプロトコル層に伝送されるように、ＲＰＣ／ＩＰＣ機構を使用してネットワーク・サーバに送られる。ネットワーク・サーバがNetworkOperationオブジェクトを受け取ると、それは操作オブジェクト上でExecute()メソッドを呼び出し、次に適切なプロトコル・インプリメンテーション層オブジェクト上で対応する関数を呼び出す。

【０１６９】したがって、TNetworkOperationオブジェクトは、それに対して要求したタスクを実行するために「組込み知能」を備えている。ネットワーク・サーバは、TNetworkOperationオブジェクトを受け取ると、そのオブジェクト内のExecute()メソッドを呼び出す。クライアント要求の特徴とは無関係に、サーバ機能は同じである。各クライアント要求用のExecute()メソッドは、クライアント要求を適切なプロトコル層オブジェクトに伝送するために必要な情報をすべて含んでいる。

【０１７０】TNetworkOperationオブジェクトは、TProtocolInterfaceの具象クラスでも作成することができる。たとえば、TBindOpのデフォルト・クラスがＴＣＰ／ＩＰの要求を満足しない場合、ＴＣＰ／ＩＰインタフェースは、TTCPBindOpクラスを作成するようにTBindOpを指定変更する。その後、クライアントがバインド要求を行うと、具象ＴＣＰ／ＩＰクライアントＡＰＩを表すTTCPINFクラスはTTCPBindOpオブジェクトを作成する。特定のクライアント要求の意味を再定義するか、または新しい要求とその操作オブジェクトを追加することをTNetworkOperationクラスから継承できる能力により、この方式は極めて柔軟かつ強力なものになる。

【０１７１】図２１は、特定のクライアント要求を満たすために作成されるクラス・オブジェクトのクラス階層を示している。同図のTTCPBindOpオブジェクト７１５は、クライアント・アプリケーションからのバインド要求に対応するTTCPTINF::Bind()メソッドによって作成される。これにより、TProtocolInterface::Bind()メソッドが指定変更される。TTCPNew1Op７１７とTTCPNew2Op７１９は、いくつかのクライアント要求の場合にＴＣＰ／ＩＰに固有の２つの新しい操作オブジェクトの例である。

【０１７２】２．TNetworkOperationの関数

TNetworkOperationクラスが提供する重要な関数の一部を以下に示す。

【０１７３】Classコンストラクタ：　この関数は、クライアント・アプリケーションによる要求の対象となるプロトコル層インデックスを設定するものである。

【０１７４】層インデックスは、その要求をプロトコル・スタックのトランスポート層、ネットワーク層、ファミリー層のいずれに送るべきかを識別するものである。

【０１７５】Execute()：　この関数は、TNetworkOperationオブジェクトを受け取ったときにサーバによって呼び出される。この関数は、操作オブジェクトから層インデックスを獲得し、操作オブジェクトから関連パラメータを収集し、スタックの適切なTProtocolLayerオブジェクトへの呼出しを行うものである。たとえば、TBindOp::Execute()は、TTCPProtocolAddressオブジェクトをBind()関数へのパラメータとしてTProtocolLayer::Bind()を呼び出すはずである。

【０１７６】Get/SetLayerIndex：　この関数は、操作オブジェクトで層インデックスを返す／設定するものである。

【０１７７】SetLocationToClient：　デフォルトでは、操作オブジェクトがその位置をサーバに設定する。操作オブジェクトは、サーバ上にあるか、クライアント上にあるかによって、挙動が異なってくる。たとえば、TProtocolLayer::Bind()関数へのパラメータなので、TBindOpはTNetworkAddressオブジェクトをサーバに送る必要がある。しかし、サーバはアドレスをクライアントに戻す必要はない。位置フラグを使用して、パラメータの受渡しが制御される。ネットワーク操作オブジェクトがクライアントによって作成されると、そのオブジェクトは位置をクライアントに設定する。

【０１７８】Stream-out演算子：　この関数は、操作オブジェクトを平坦化し、そのオブジェクトのデータ・メンバーをデータ・ストリームに入れるために使用する。たとえば、TBindOp用のStream-out演算子は、それがクライアントからのオブジェクトを送信する場合にTNetworkAddressオブジェクトを平坦化する。TSendOpは、バッファおよび宛先アドレスをサーバに送るためにバッファ・アドレスとTNetworkAddressオブジェクトを平坦化することができる。次にサーバはSendOp::Execute()を呼び出し、次にこれがユーザ・データとともにTProtocolLayer::Xmit()メソッドを呼び出して、宛先にデータを送る。送信が完了すると、サーバはＲＰＣ／ＩＰＣ機構を使用してTSendCompletionオブジェクトをクライアントに戻す。TSendOp用のStream-out演算子は、それがサーバであるかどうかを検査し、TSendCompletionオブジェクトをストリームアウトする。

【０１７９】Stream-in演算子：　この関数は、オブジェクトが平坦化された場合にデータ・ストリームからオブジェクトを再構築するために使用する。Stream-out演算子とは逆の操作であることは明らかである。操作オブジェクトは、データ・ストリームにオブジェクトを平坦化したり、データ・ストリームからオブジェクトを再構築するために位置フラグを使用する。

【０１８０】３．クライアントサーバ通信

ユーザが作成するどの通信端点についても、プロトコル層オブジェクトのスタックに端点を関連付けるために維持しなければならない固有のＩＤが存在していなければならない。プロトコル層オブジェクトは、サーバ・プロセス上の端点を表す。ただし、この対応は１対１であることに留意されたい。このため、TAccessDefinition::Instantiate()メソッドを使用して端点の作成中にTAccessOpオブジェクトが作成される。TAccessOpオブジェクトは、通信のクライアント側を表すClientStackHeadオブジェクトを作成する。次にTAccessOpオブジェクトは平坦化され、ClientStackHeadによりＲＰＣまたはＩＰＣ機構を使用してサーバに送られる。次にサーバは、TNetworkOperationのstream-in演算子を使用してデータ・ストリームからTAccessOpオブジェクトを再構築し、TAccessOp::Execute()メソッドを呼び出す。この関数は、プロトコル・インタフェース・オブジェクトからプロトコル層オブジェクトを作成するServerStackHeadオブジェクトを作成し、TServerStackHeadオブジェクトでこれらのプロトコル層オブジェクトへのポインタのリストを保持する。TServerStackHeadポインタはネットワーク・サーバのグローバル・テーブルに格納され、インデックスはクライアントにストリームアウトされる。TClientStackHeadオブジェクトは、ServerStackHeadIDを格納し、後続のすべての操作にそれを使用する。したがって、ServerStackHeadIDは、クライアントとサーバとの間で固有のＩＤとして機能する。TBindOpなどの後続の要求は、サーバによって受け取られると、TBindOpで渡されるＩＤを使用して、対応するサーバ・スタック・ヘッドの位置を特定する。

【０１８１】TClientStackHeadとTServerStackHeadは、所与の端点についてクライアントとサーバとのやりとりを管理する。これらのオブジェクトは、クライアント端点のハンドルであるTAccessDefinitionオブジェクトと、サーバ上の端点を表すTProtocolLayerオブジェクトの対応スタックとの間をリンクする。また、ClientStackHeadとServerStackHeadの対は、クライアントとサーバをリンクする内部管理オブジェクトを構成する。

【０１８２】TClientStackHeadの重要な関数の一部を以下に示す。

【０１８３】１．ProcessOperation：　この関数は、TClientStackHeadがTNetworkOperationオブジェクトを処理するために、TProtocolInterfaceまたはTAccessDefinitionオブジェクトによって呼び出される。この関数は、TNetworkOperationオブジェクトを平坦化し、システムが提供するＲＰＣ／ＩＰＣ機構を使用して、平坦化した操作オブジェクトをサーバに送る。また、この関数は、送られた要求に対してサーバが応答したときにNetworkOperationオブジェクトも再構築する。基本的にこの関数は、サーバとの間でNetworkOperationオブジェクトを送受信するものである。

【０１８４】２．SaveServerInfo：　このメソッドは、ServerStackHeadIDをClientStackHeadに保管するために呼び出されるものである。AccessDefinitionがインスタンス化されると、サーバは端点用に作成されたServerStackHead用のＩＤとともにTAccessOpオブジェクトを返す。その後、サーバに対して要求が送られるときは、必ずNetworkOperationがこのＩＤを使用する。

【０１８５】３．CancelRequests：　この関数は、クライアント・アプリケーションによってTProtocolInterface::CancelRequestsが呼び出されたとき、またはクライアント・アプリケーションが異常終了したときに呼び出されるものである。ClientStackHeadは、必要な終結処置を行うようServerStackHeadに通知するCancelRequestsOp操作オブジェクトを作成する。

【０１８６】ServerStackHeadの重要な関数の一部を以下に示す。

【０１８７】１．Classコンストラクタ：　このコンストラクタは、TProtocolInterfaceオブジェクトのスタックを受け取り、対応するTProtocolLayerオブジェクトを構築する。これは、後続の要求からこれらの層オブジェクトを呼び出すためにこれらの層オブジェクトへのポインタを維持している。

【０１８８】２．ProcessOperation：　この関数は、クライアントからNetworkOperationオブジェクトを受け取ったときにNetworkServerProcessによって呼び出されるものである。この関数は、適切なProtocolLayerオブジ

ェクトの位置を特定した後、TNetworkOperationオブジェクト上でExecute()メソッドを呼び出す。次にExecute()メソッドは、ProtocolLayerオブジェクト上で必要なメソッドを呼び出す。

【０１８９】３．GetProtocolLayer：　この関数は、インデックスが付けられたTProtocolLayerオブジェクトへのポインタを返すものである。このインデックスはクライアント・アプリケーションによって渡される。

【０１９０】４．CancelRequests：　この関数は、プロトコル・スタック上のすべての保留要求を取り消すためにTCancelRequestsOpを受け取ったときに呼び出されるものである。これは、TClientStackHeadのCancelRequestsに対応するサーバ側の関数である。

【０１９１】図２２は、上記のオブジェクトのクラス階層を示し、Ｂｏｏｃｈの表記法を使用してオブジェクト・モデルを記述している。TAccessDefinition１０１は、TNetworkDefinition１００から派生したものである。TAccessDefinition１０１は、端点を構成する様々なTProtocolInterface１３５オブジェクトを含んでいる。また、TAccessDefinition１０１は、クライアントサーバ通信のプリミティブ機能のすべてを実行するTClientStackHead７２１も含んでいる。TServerStackHead７２３は、ClientStackHeadに対応するサーバ側のクラスである。TClientStackHeadとTServerStackHeadの対は、プロトコル・インタフェースとプロトコル・インプリメンテーション層オブジェクトとのリンクを表し、したがって、クライアント・アプリケーションとサーバ上のプロトコル・スタックとをリンクする。TProtocolInterfaceクラス１３５とTProtocolLayerクラス１５１はともに共通基本クラスであるMProtocolService１３３から派生したものである。クライアントからのネットワーク要求はいずれもTNetworkOperation７０１オブジェクトで折り返されるサーバに送られる。すなわち、TProtocolInterface内のメソッドが適切なTNetworkOperationオブジェクトを作成し、操作オブジェクトはTClientStackHeadによって平坦化され、サーバに送られる。TNetworkOperationオブジェクトは、TClientStackHeadとTServerStackHeadの両方を使用する。

【０１９２】図２３は、クライアントからサーバへの要求とその逆の要求の流れを示している。前述のように、端点は、クライアント上のTAccessDefinitionオブジェクトと、サーバ上のTProtocolLayerオブジェクトのスタックによって表される。クライアントとサーバとのやりとりは、TClientStackHeadオブジェクトとTServerStackHeadオブジェクトによって管理される。同図は２つの端点７２５を示している。クライアントからのネットワーク要求は、システムのＲＰＣまたはＩＰＣ機構を使用してTNetworkOperationオブジェクトとしてServerProcess/Thread７２７に送られる。ServerProcessは、TNetworkOperationオブジェクトを再構築し、次に端点１を表すTServerStackHeadオブジェクトの位置を特定し、ServerStackHead１７２９に要求を経路指定する。サーバ上のこの端点を表すServerStackHead2へ端点２から要求を経路指定する場合も同様の処理が行われる。次に、ServerStack1はTNetworkOperation::Execute()メソッドを呼び出し、これがスタック１　７３３のTTransportLayerの適切なメソッドを呼び出す。TTransportLayerは、要求をTFamilyLayer７３７に送るTNetworkLayerのメソッドの呼出しを選ぶこともできる。次にファミリー層は、スタック１　７３３のTDataLinkLayerに要求を送る。TTransportLayerは、要求をTFamilyLayer７３７に送るTNetworkLayerのメソッドの呼出しを選ぶこともできる。次にファミリー層は、要求の処理後、アダプタに要求を送ることができるTDataLinkLayer７３９に要求を送る。DataLinkLayerは、アダプタを介して何らかのパケットを受信すると、そのパケットをファミリー層にディスパッチする。TFamilyLayerは、そのパケットを適切なスタックに経路指定する。呼出しからTTransportLayerに戻ると、TNetworkOperation::Execute()は応答を収集し、システムのＲＰＣ／ＩＰＣ機構を使用して適切なTClientStackHeadにTNetworkOperationオブジェクトを戻す。この手続きは、サーバ上の端点を表す端点２とスタック２　７３５上のいかなる要求についても同じである。

【０１９３】図２３は、クライアントとサーバのクラス・オブジェクト間の関係を示している。以下の例では、一般的なネットワークのクライアント／サーバ・モデルを想定している。

【０１９４】３．例

ＴＣＰ／ＩＰトランスポート・インタフェースを表すTTCPTINFについて検討する。TTCPTINF::Bind()が入力パラメータとしてTTCPAddressを取り、TTCPAddressオブジェクトを返すものと想定する。ただし、TTCPAddressはTProtocolAddressから派生したものであることに留意されたい。図２４に示すように、TTCPTINF::Bind()メソッドは以下のように実行する。

【０１９５】ステップ８０１では、TNetworkOperationオブジェクトであるTTCPBindOpが作成される。

【０１９６】ステップ８０３では、このAccessDefinition用のClientStackHeadオブジェクトを獲得する。

【０１９７】ステップ８０５では、TTCPBindOpオブジェクトのTTCPAddressを平坦化し、要求をサーバに送るために、ClientStackHead::ProcessOperation()が呼び出される。サーバは、ステップ８０７でＲＰＣ／ＩＰＣなどのメッセージ・ストリームから平坦化したTTCPBindOpを再構築する。

【０１９８】ステップ８０９では、サーバは、TTCPBindOpオブジェクトで渡されたスタックＩＤからServerStackHeadオブジェクトの位置を特定する。

【０１９９】ステップ８１１では、ServerStackHeadオブジェクトが適切なプロトコル層オブジェクトにTProto

colLayer::Bind()を呼び出す。

【０２００】ステップ８１３では、バインドするための呼出しがTTCPAddressオブジェクトを返し、サーバがTTCPBindOpで復帰情報（アドレス）を復元し、それをクライアントにストリームで戻す。ClientStackHeadはメッセージ・ストリームからTTCPBindOp()オブジェクトを再構築する。最後に、ステップ８１７では、TTCPINF::Bind()がTTCPBindOpからTTCPAddressを取り出し、それをクライアント・プログラムに返す。

【０２０１】特定の実施例に関連して本発明を示し説明してきたが、当業者であれば、修正を加え他の環境で本発明を実施できることを理解できるだろう。たとえば、上記の本発明はソフトウェアによって選択的に再構成または活動化される汎用コンピュータで都合よく実施することができるが、当業者は、上記の発明を実行するために特別に設計された専用装置を含む、ハードウェア、ファームウェア、またはソフトウェアとファームウェアとハードウェアの組合せで本発明を実施できることに留意されたい。したがって、特許請求の範囲に記載する本発明の精神および範囲を逸脱せずに、形式および細部の変更を行うことができる。

【０２０２】まとめとして、本発明の構成に関して以下の事項を開示する。

【０２０３】（１）サーバ・システム内の実行ユニットの、クライアント・プロセスとサーバ・プロセスとの間の通信プロセス専用のプールを動的に管理するための方法において、通信プロセス・プール内に、典型的なクライアント・ロードをサポートするのに必要な数である最小数、およびサーバ・システムにとって過負荷にならずにピーク・クライアント・ロードをサポートするための上限である最大数の実行ユニットを割り振るステップと、クライアントによるサービス要求をサーバ・システムが受信するステップとを含み、受信した各クライアント要求ごとに、受信したクライアント要求に実行ユニットを割り当てると、通信プロセス・プール内の現行数の実行ユニットが最大数の実行ユニットを上回ることになるかどうかを判定し、上回る場合にクライアント要求を拒否するステップと、受信したクライアント要求に実行ユニットを割り当てると、その要求を行うクライアント・タスクに割り当てられた現行数の実行ユニットがクライアント・タスク用の割り当てられた実行ユニットの数を上回ることになるかどうかを判定し、上回る場合にクライアント要求を拒否するステップと、判定ステップが否定であればクライアント要求を許可し、その結果、通信プロセス・プール内の実行ユニットがクライアント要求に割り当てられるステップとを含むことを特徴とする方法。

（２）サーバ・システム内の実行ユニットの、クライアント・プロセスとサーバ・プロセスとの間の通信プロセス専用のプールを動的に管理するためのシステムにおいて、通信プロセス・プール内に、典型的なクライアント・ロードをサポートするのに必要な数である最小数、およびサーバ・システムにとって過負荷にならずにピーク・クライアント・ロードをサポートするための上限である最大数の実行ユニットを割り振る手段と、クライアントによるサービス要求をサーバ・システムが受信する手段と、受信したクライアント要求に実行ユニットを割り当てると、通信プロセス・プール内の現行数の実行ユニットが最大数の実行ユニットを上回ることになるかどうかを判定する手段と、受信したクライアント要求に実行ユニットを割り当てると、その要求を行うクライアント・タスクに割り当てられた現行数の実行ユニットがクライアント・タスク用の割り当てられた実行ユニットの数を上回ることになるかどうかを判定する手段と、通信プロセス・プール内の実行ユニットをクライアント要求に割り当てることができると判定手段が確認した場合にクライアント要求を許可する手段とを含むことを特徴とするシステム。

（３）サーバ・システム内の実行ユニットの、クライアント・プロセスとサーバ・プロセスとの間の通信プロセス専用のプールを動的に管理するためのコンピュータが読取り可能な媒体上のコンピュータ・プログラム製品において、通信プロセス・プール内に、典型的なクライアント・ロードをサポートするのに必要な数である最小数、およびサーバ・システムにとって過負荷にならずにピーク・クライアント・ロードをサポートするための上限である最大数の実行ユニットを割り振るようにシステムに指示する手段と、クライアントによるサービス要求をサーバ・システムが受信するようにシステムに指示する手段と、受信したクライアント要求に実行ユニットを割り当てると、通信プロセス・プール内の現行数の実行ユニットが最大数の実行ユニットを上回ることになるかどうかを判定するようにシステムに指示する手段と、受信したクライアント要求に実行ユニットを割り当てると、その要求を行うクライアント・タスクに割り当てられた現行数の実行ユニットがクライアント・タスク用の割り当てられた実行ユニットの数を上回ることになるかどうかを判定するようにシステムに指示する手段と、通信プロセス・プール内の実行ユニットをクライアント要求に割り当てることができるという判定に応答してクライアント要求を許可するようにシステムに指示する手段とを含むことを特徴とするコンピュータ・プログラム製品。

【図面の簡単な説明】

【図１】本発明の教示により構成されたコンピュータ・システムを示す図である。

【図２】ネットワーク定義オブジェクト用のクラス階層を示す図である。

【図３】ネットワーク・アドレス・クラス用のクラス階層を示す図である。

53

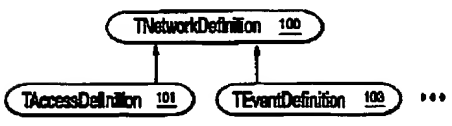
【図４】プロトコル・インタフェース・クラス用のクラス階層を示す図である。
【図５】プロトコル層クラス用のクラス階層を示す図である。
【図６】接続指向遷移の状態図である。
【図７】無接続状態遷移の状態図である。
【図８】本発明のＴＣＰ／ＩＰ実施例のクラス関係を示す図である。
【図９】本発明によりネットワーク接続をセットアップするためのプロセスの流れ図である。
【図１０】図９に示すネットワーク接続プロセスの専用ネットワーク層にある様々なオブジェクトの体系図である。
【図１１】ネットワーク事象オブジェクト用のクラス階層を示す図である。
【図１２】単一の通信端点から事象を収集するためのプロセスの流れ図である。
【図１３】複数の通信端点から事象を収集するためのプロセスの流れ図である。
【図１４】ネットワーク・プロトコル・サーバのクライアント要求を処理するために通信スレッドのプールを管理するための第１および第２の実施例の体系図である。
【図１５】ネットワーク・プロトコル・サーバのクライアント要求を処理するために通信スレッドのプールを管理するための第１および第２の実施例の体系図である。
【図１６】通信スレッドのプール用の管理プロセスの流れ図である。
【図１７】通信スレッドのプール用の管理プロセスの流れ図である。
【図１８】通信スレッドのプール用の管理プロセスの流れ図である。
【図１９】通信スレッドのプール用の管理プロセスの流れ図である。
【図２０】ネットワーク操作オブジェクト用のクラス階層図である。

54

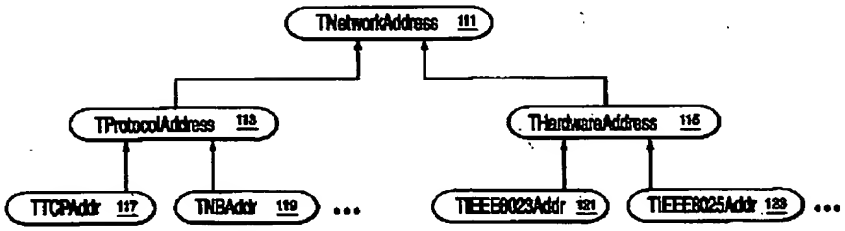
【図２１】ネットワーク操作オブジェクト用のクラス階層図である。
【図２２】本発明の様々なクラス間のクラス関係を示す図である。
【図２３】本発明によるネットワーク内の様々なオブジェクト間のメッセージの流れを示す図である。
【図２４】ネットワーク操作オブジェクトでネットワーク・プロトコル要求を渡すための流れ図である。
【符号の説明】

１１　システム・ユニット
１２　キーボード
１３　マウス
１４　グラフィック・ディスプレイ
１５Ａ　スピーカ
１５Ｂ　スピーカ
２１　システム・バス
２２　マイクロプロセッサ
２３　ＲＯＭ
２４　ＲＡＭ
２５　メモリ管理チップ
２６　ハード・ディスク・ドライブ
２７　フロッピー・ディスク・ドライブ
２８　キーボード制御装置
２９　マウス制御装置
３０　ビデオ制御装置
３１　オーディオ制御装置
３２　ＣＤ－ＲＯＭ
３３　ディジタル信号プロセッサ
４０　入出力制御装置
４６　ネットワーク
４８　オペレーティング・システム
４９　クライアント・アプリケーション
５０　クライアント・インタフェース
５１　プロトコル・オブジェクト
５２　サーバ・アプリケーション

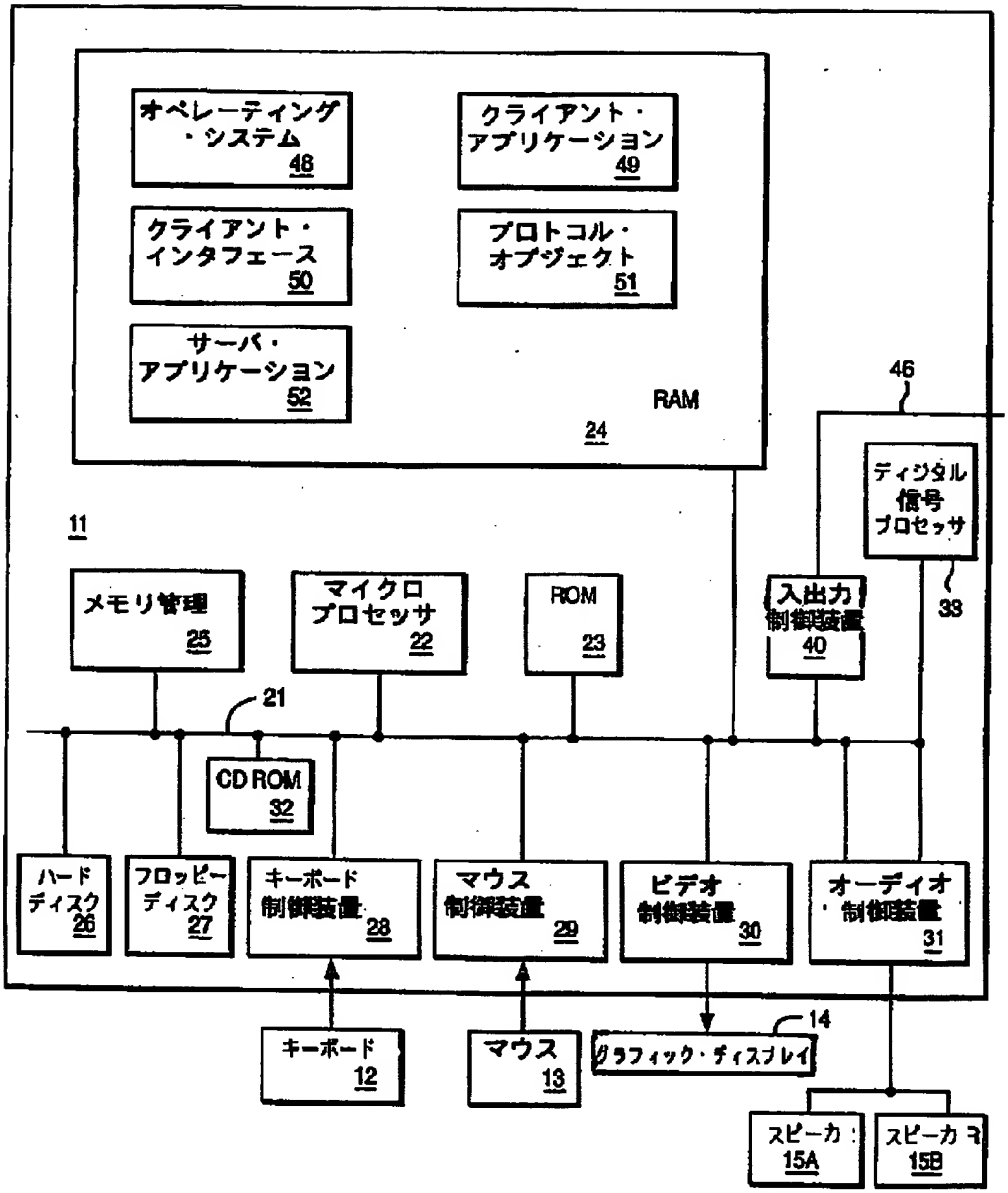
【図２】



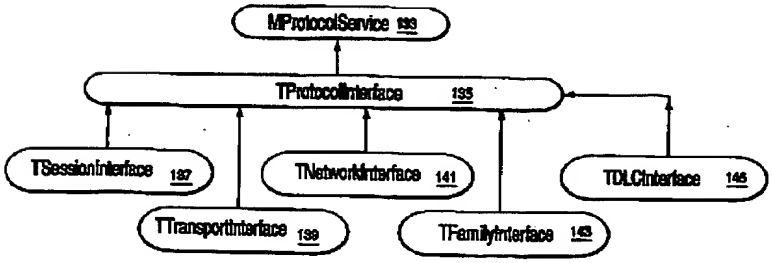
【図３】



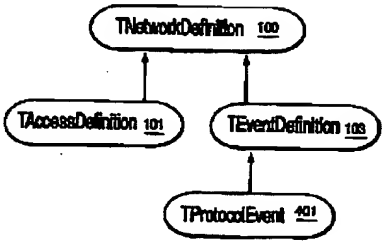
【図１】


オペレーティング・システム 48
クライアント・アプリケーション 49
クライアント・インタフェース 50
プロトコル・オブジェクト 51
サーバ・アプリケーション 52
RAM 24
46
ディジタル信号プロセッサ
33
11
メモリ管理 25
マイクロプロセッサ 22
ROM 23
入出力制御装置 40
21
CD ROM 32
ハードディスク 26
フロッピーディスク 27
キーボード制御装置 28
マウス制御装置 29
ビデオ制御装置 30
オーディオ制御装置 31
14
キーボード 12
マウス 13
グラフィック・ディスプレイ
スピーカ 15A
スピーカ 15B

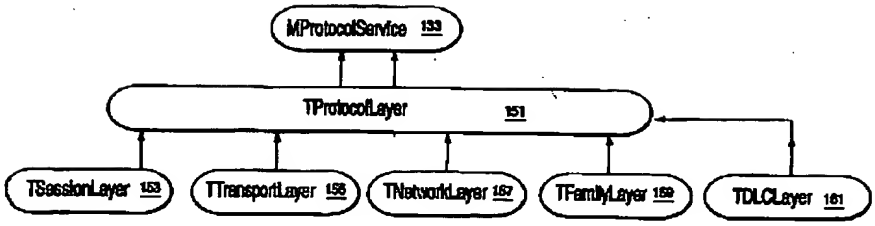
【図４】


MProtocolService 193
TProtocolInterface 135
TSessionInterface 137
TNetworkInterface 141
TDLCInterface 145
TTransportInterface 139
TFamilyInterface 143

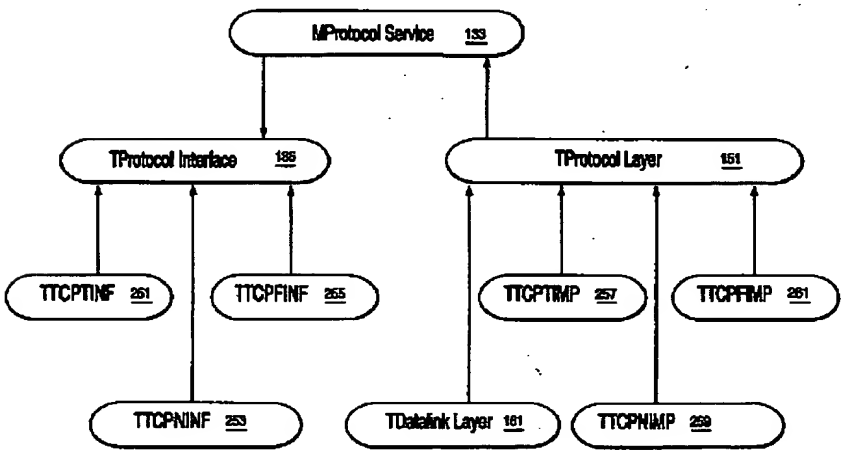
【図１１】


TNetworkDefinition 100
TAccessDefinition 101
TEventDefinition 103
TProtocolEvent 401

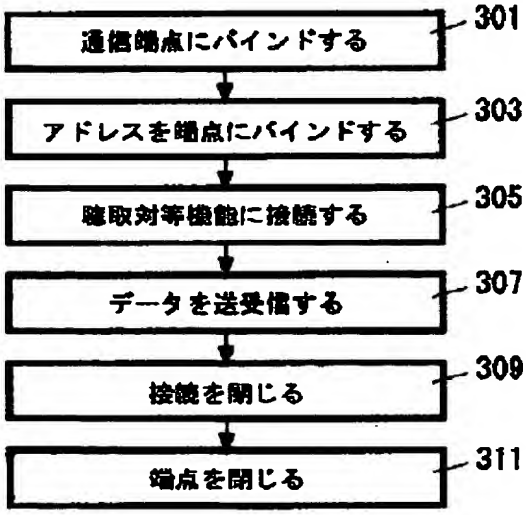
【図５】



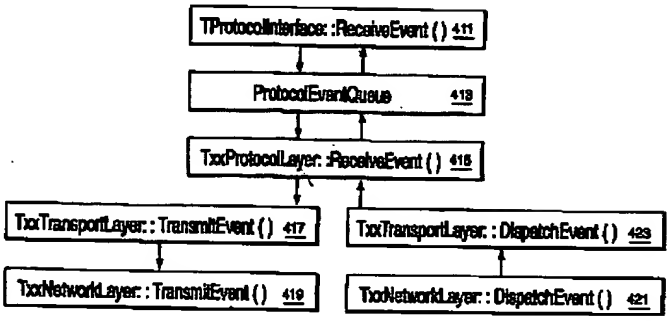
【図８】



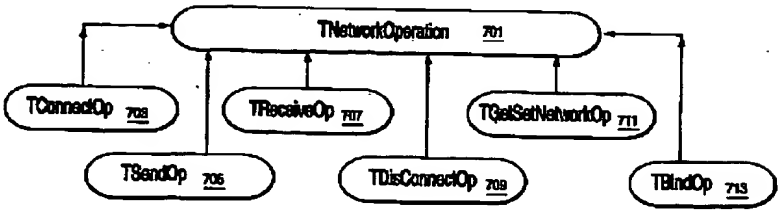
【図９】



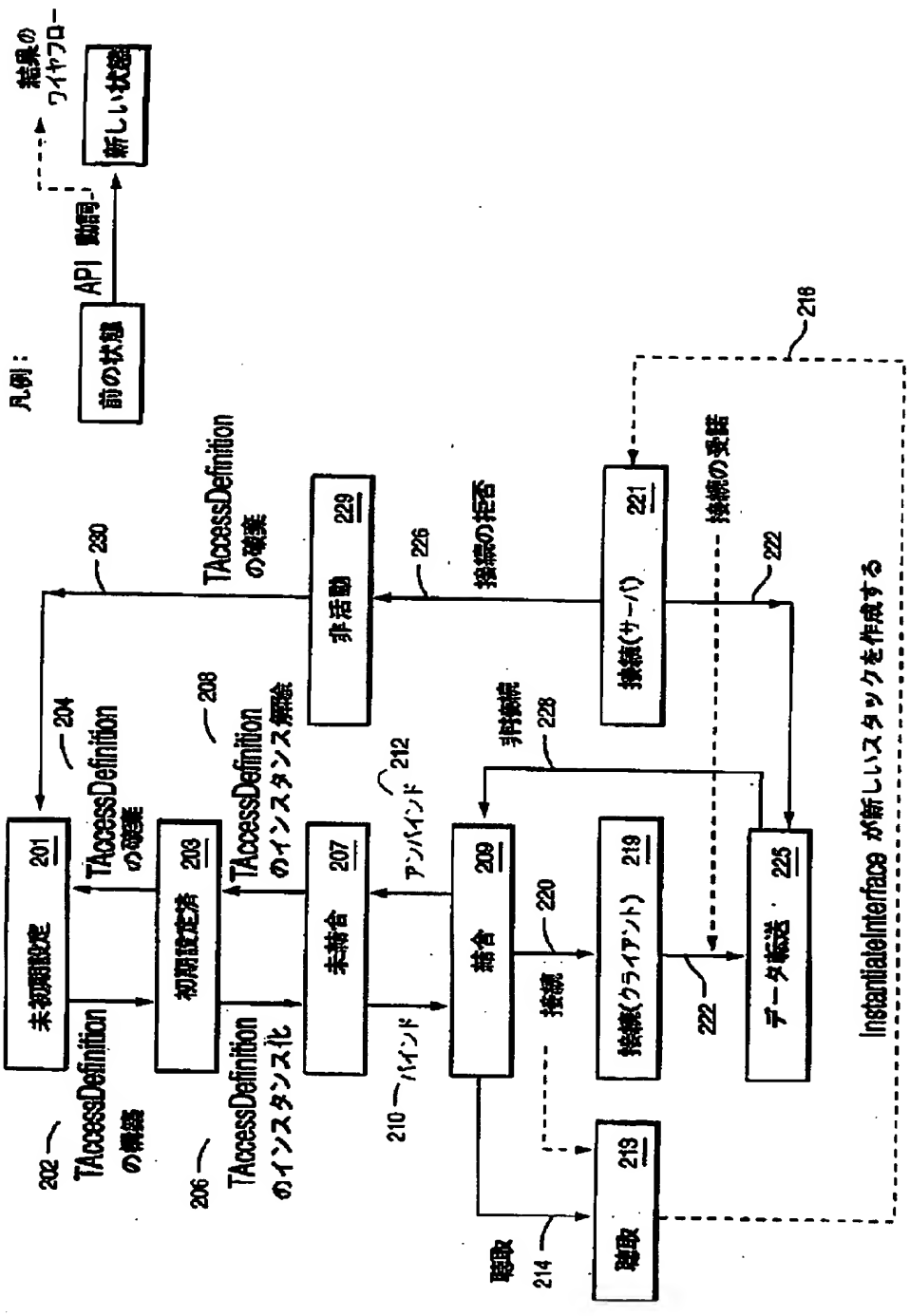
【図１２】



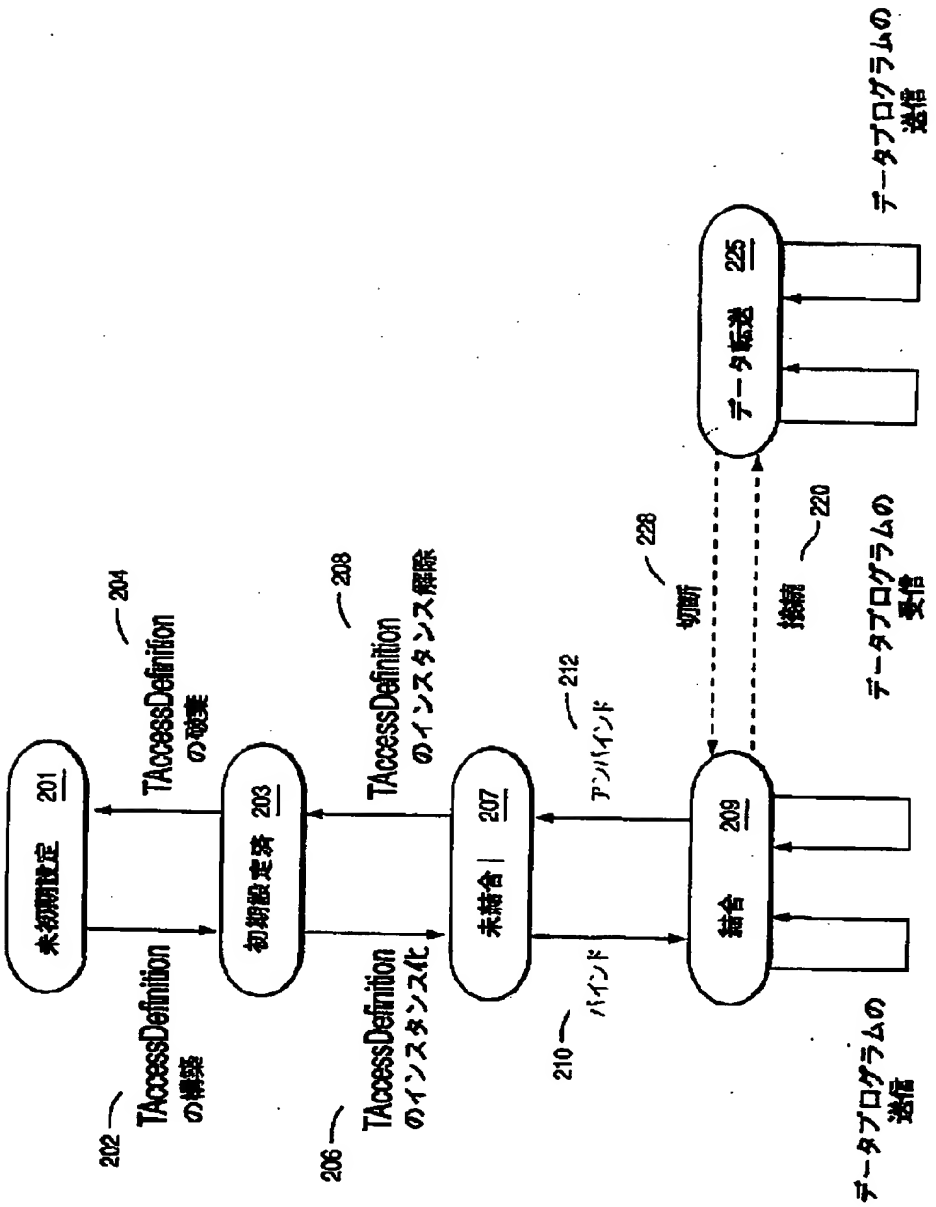
【図２０】



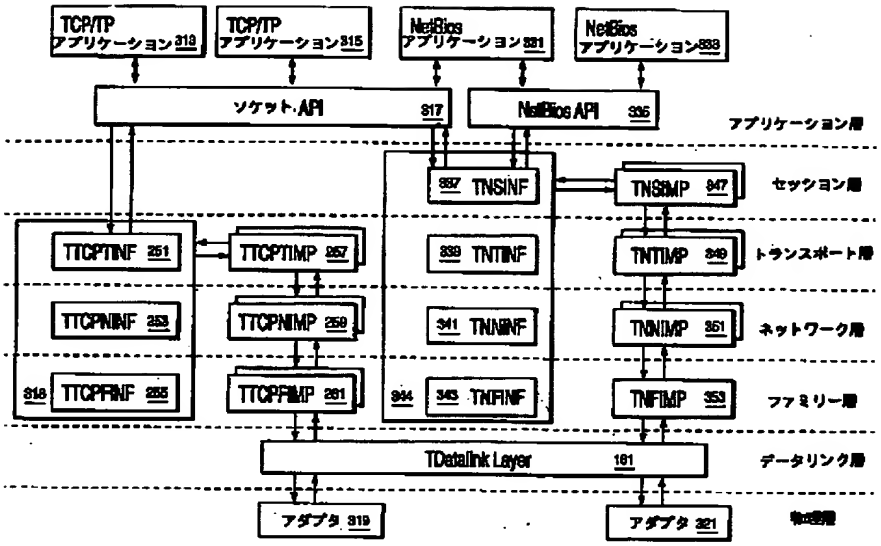
【図６】


凡例：
前の状態
API 動詞
新しい状態
結果のワイヤフロー
TAccessDefinition の確立
非活動 229
接続の拒否
接続(サーバ) 221
接続の受諾
216
230
204
208
222
226
非接続
228
未初期設定 201
TAccessDefinition の確立
初期設定済 203
TAccessDefinition のインスタンス解除
未結合 207
アンバインド 212
結合 209
接続(クライアント) 219
データ転送 225
220
222
TAccessDefinition の構築
TAccessDefinition のインスタンス化
バインド
接続
InstantiableInterface が新しいスタックを作成する
202
206
210
聴取
214
聴取 213

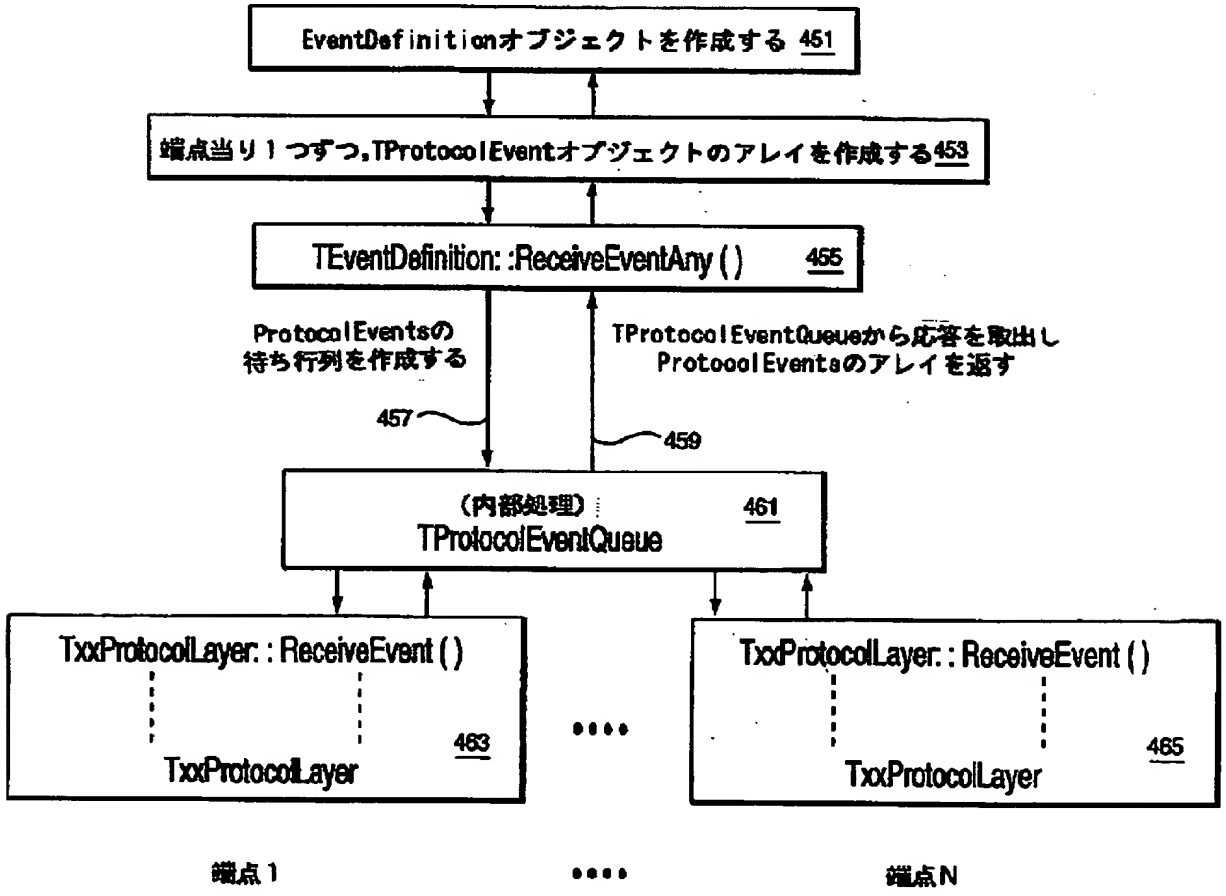
【図７】


未初期設定 201
初期設定済 203
未結合 207
結合 209
データ転送 225
TAccessDefinition の構築
202
TAccessDefinition の破棄
204
TAccessDefinition のインスタンス化
206
TAccessDefinition のインスタンス解除
208
バインド
210
アンバインド
212
データプログラムの送信
データプログラムの受信
接続
220
切断
228

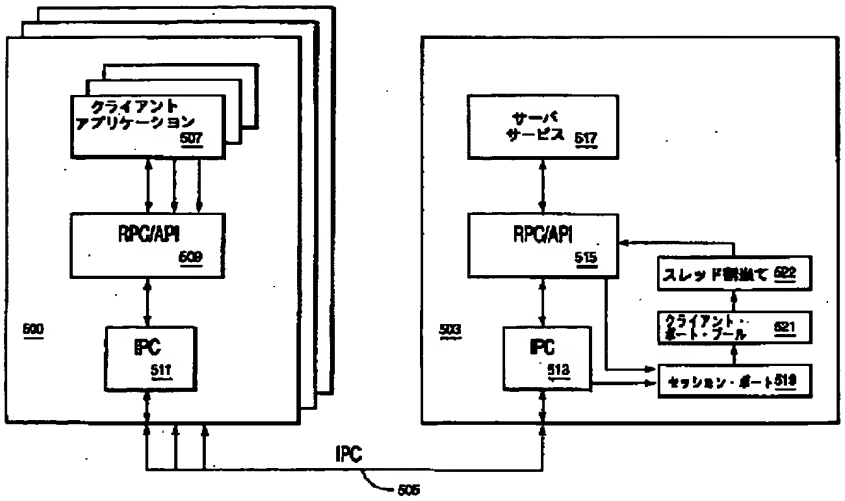
【図１０】


TCP/TP アプリケーション 313
TCP/TP アプリケーション 315
NetBios アプリケーション 331
NetBios アプリケーション 333
ソケット API 317
NetBios API 335
アプリケーション層
337 TNSINF
TNSIMP 347
セッション層
TTCPTINF 251
TTCPTIMP 257
339 TNTINF
TNTIMP 349
トランスポート層
TTCPNINF 253
TTCPNIMP 259
341 TNNINF
TNNIMP 351
ネットワーク層
318 TTCPFINF 255
TTCPFIMP 261
344 345 TNFINF
TNFIMP 353
ファミリー層
TDataLink Layer 161
データリンク層
アダプタ 319
アダプタ 321

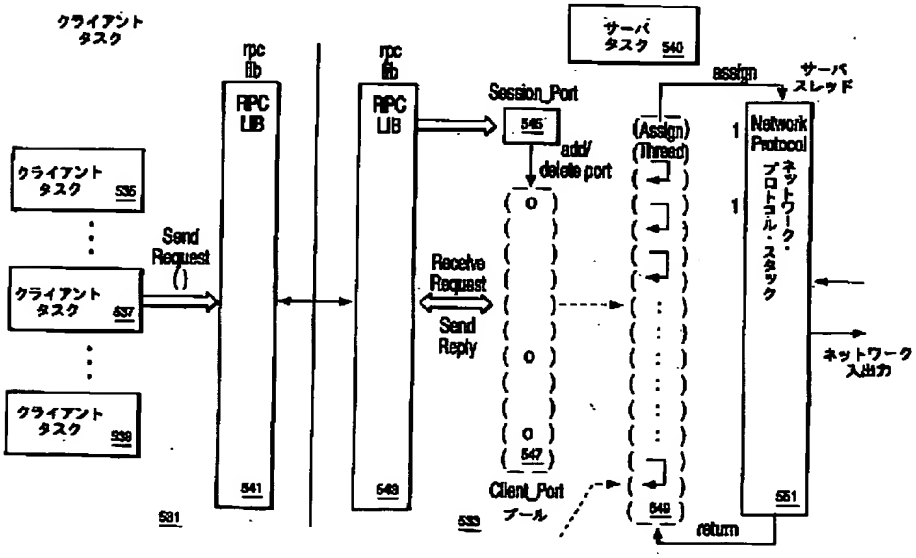
【図１３】


EventDefinitionオブジェクトを作成する 451
端点当り１つずつ,TProtocolEventオブジェクトのアレイを作成する453
TEventDefinition: :ReceiveEventAny ( ) 455
ProtocolEventsの
待ち行列を作成する
TProtocolEventQueueから応答を取出し
ProtocolEventsのアレイを返す
457
459
(内部処理) 461
TProtocolEventQueue
TxxProtocolLayer: : ReceiveEvent ( )
463
TxxProtocolLayer
TxxProtocolLayer: : ReceiveEvent ( )
465
TxxProtocolLayer
端点１
端点N

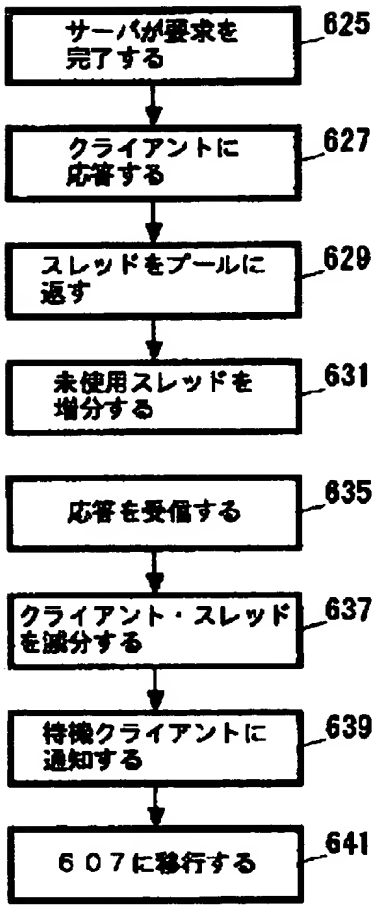
【図１４】



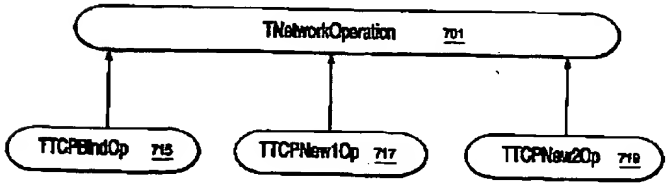
【図１５】



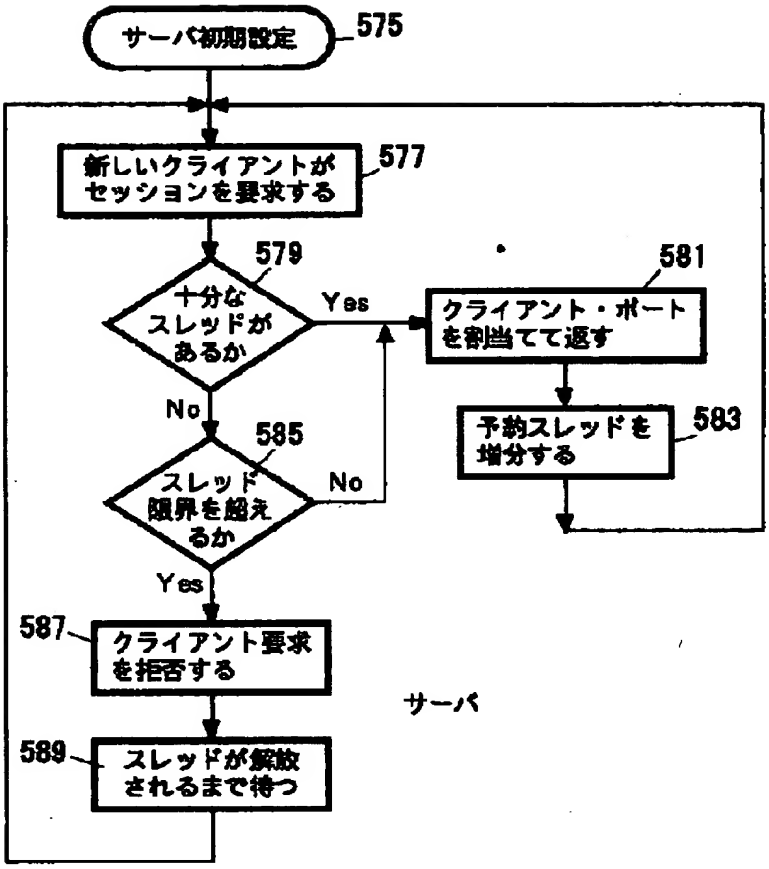
【図１８】



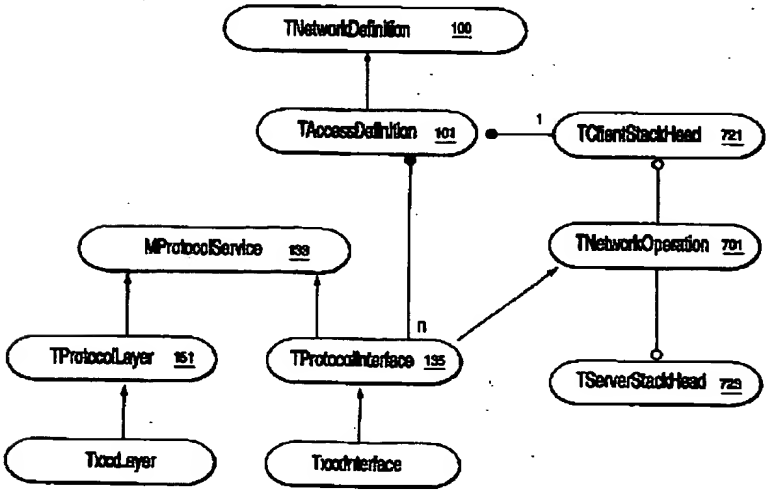
【図２１】



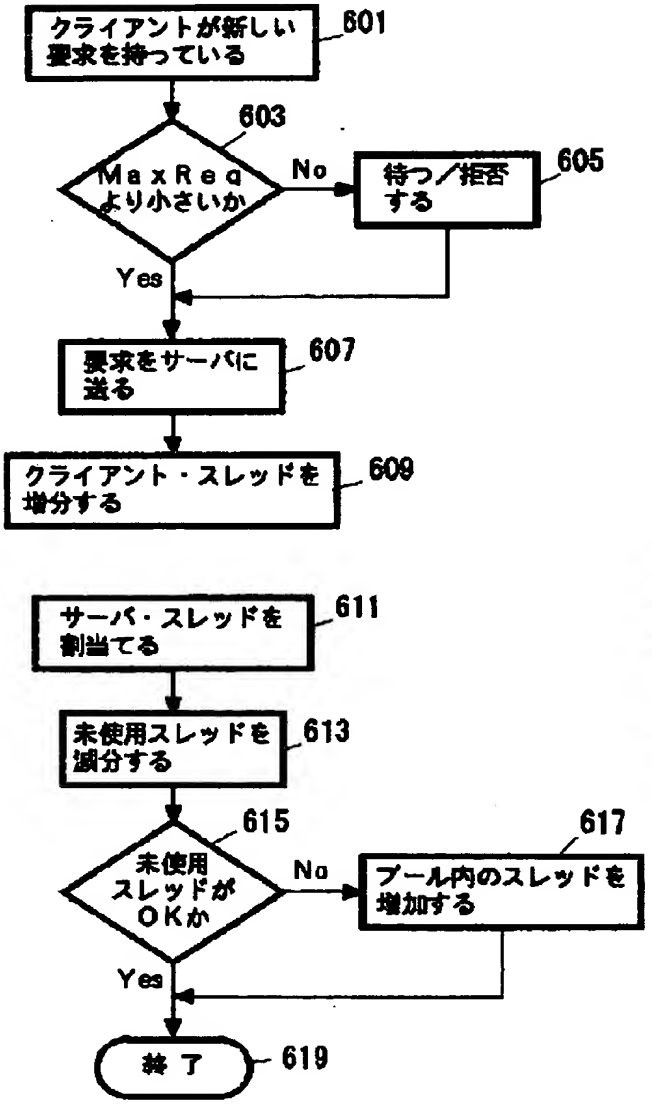
【図１６】


サーバ初期設定
575
新しいクライアントがセッションを要求する
577
十分なスレッドがあるか
579
Yes
No
クライアント・ポートを割当てて返す
581
予約スレッドを増分する
583
スレッド限界を超えるか
585
No
Yes
クライアント要求を拒否する
587
スレッドが解放されるまで待つ
589
サーバ

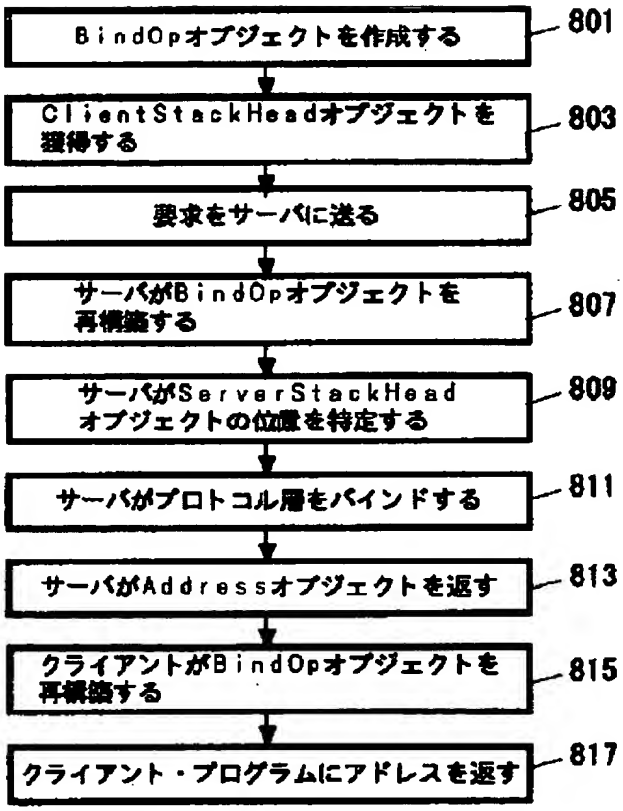
【図２２】


TNetworkDefinition 100
TAccessDefinition 101
1
TClientStackHead 721
MProtocolService 139
TNetworkOperation 701
n
TProtocolLayer 151
TProtocolInterface 135
TServerStackHead 729
TxxxLayer
TxxxInterface

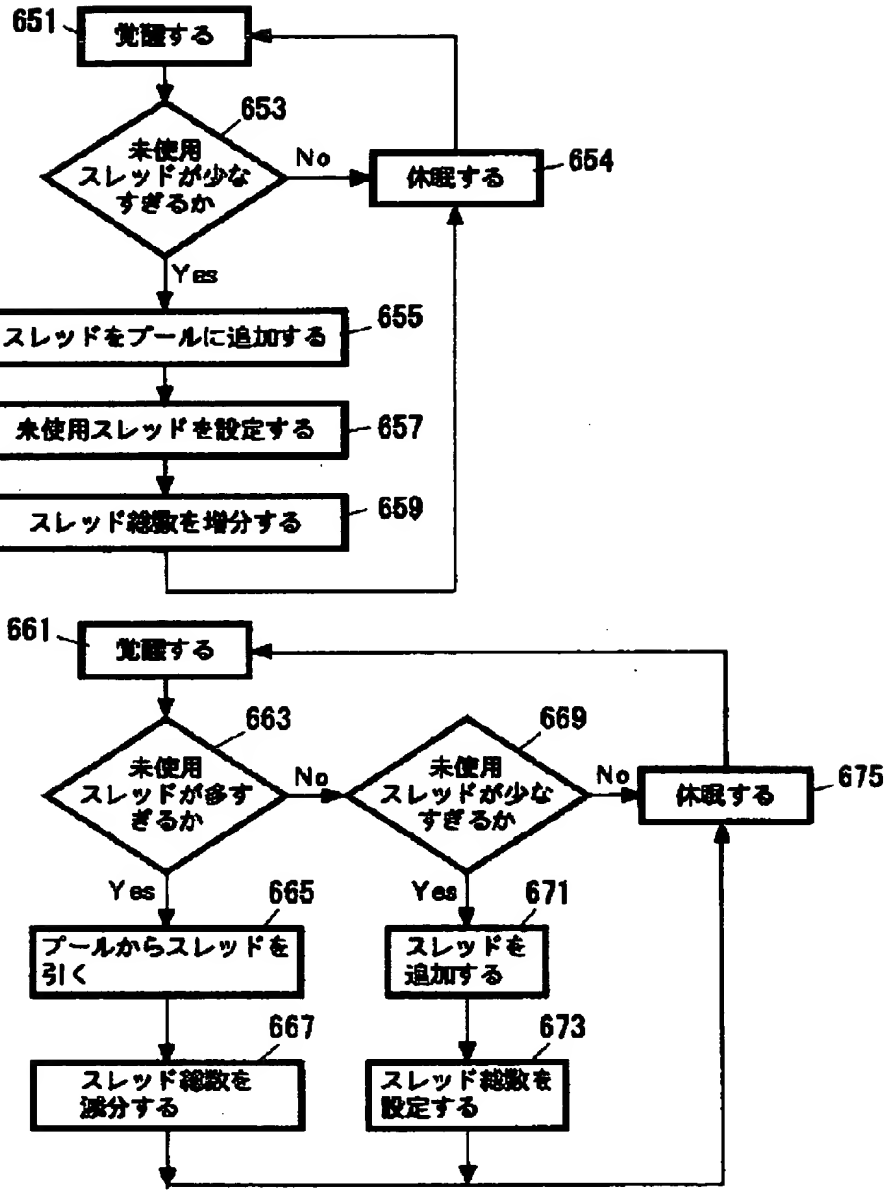
【図１７】


クライアントが新しい要求を持っている
601
603
MaxReqより小さいか
No
待つ／拒否する
605
Yes
要求をサーバに送る
607
クライアント・スレッドを増分する
609
サーバ・スレッドを割当てる
611
未使用スレッドを減分する
613
615
未使用スレッドがOKか
No
617
プール内のスレッドを増加する
Yes
終了
619

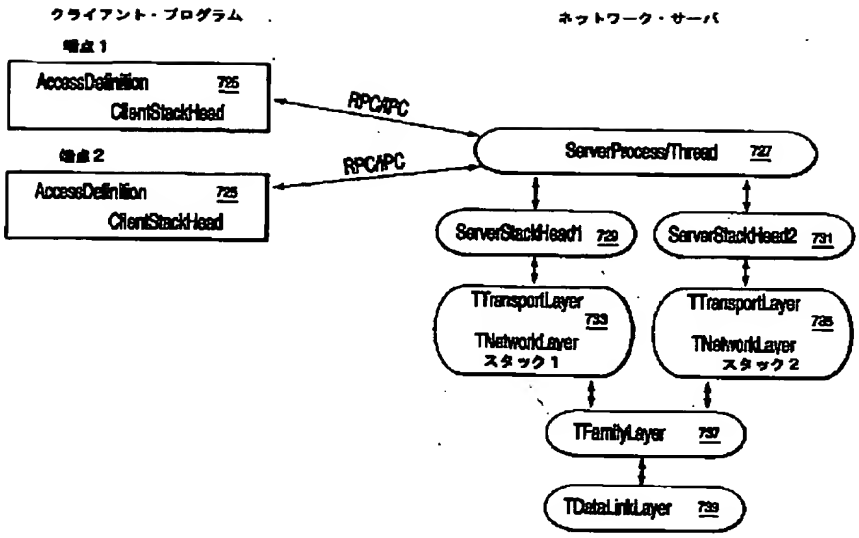
【図２４】


BindOpオブジェクトを作成する
801
ClientStackHeadオブジェクトを獲得する
803
要求をサーバに送る
805
サーバがBindOpオブジェクトを再構築する
807
サーバがServerStackHeadオブジェクトの位置を特定する
809
サーバがプロトコル層をバインドする
811
サーバがAddressオブジェクトを返す
813
クライアントがBindOpオブジェクトを再構築する
815
クライアント・プログラムにアドレスを返す
817

【図１９】


651
覚醒する
653
未使用スレッドが少なすぎるか
No
654
休眠する
Yes
655
スレッドをプールに追加する
657
未使用スレッドを設定する
659
スレッド総数を増分する
661
覚醒する
663
未使用スレッドが多すぎるか
No
669
未使用スレッドが少なすぎるか
No
675
休眠する
Yes
665
プールからスレッドを引く
667
スレッド総数を減分する
Yes
671
スレッドを追加する
673
スレッド総数を設定する

【図２３】



フロントページの続き

(72)発明者　レオ・ユエ・タク・ユング
アメリカ合衆国78759　テキサス州オース
チン　ディーケイ・ラーンチ・ロード
11714